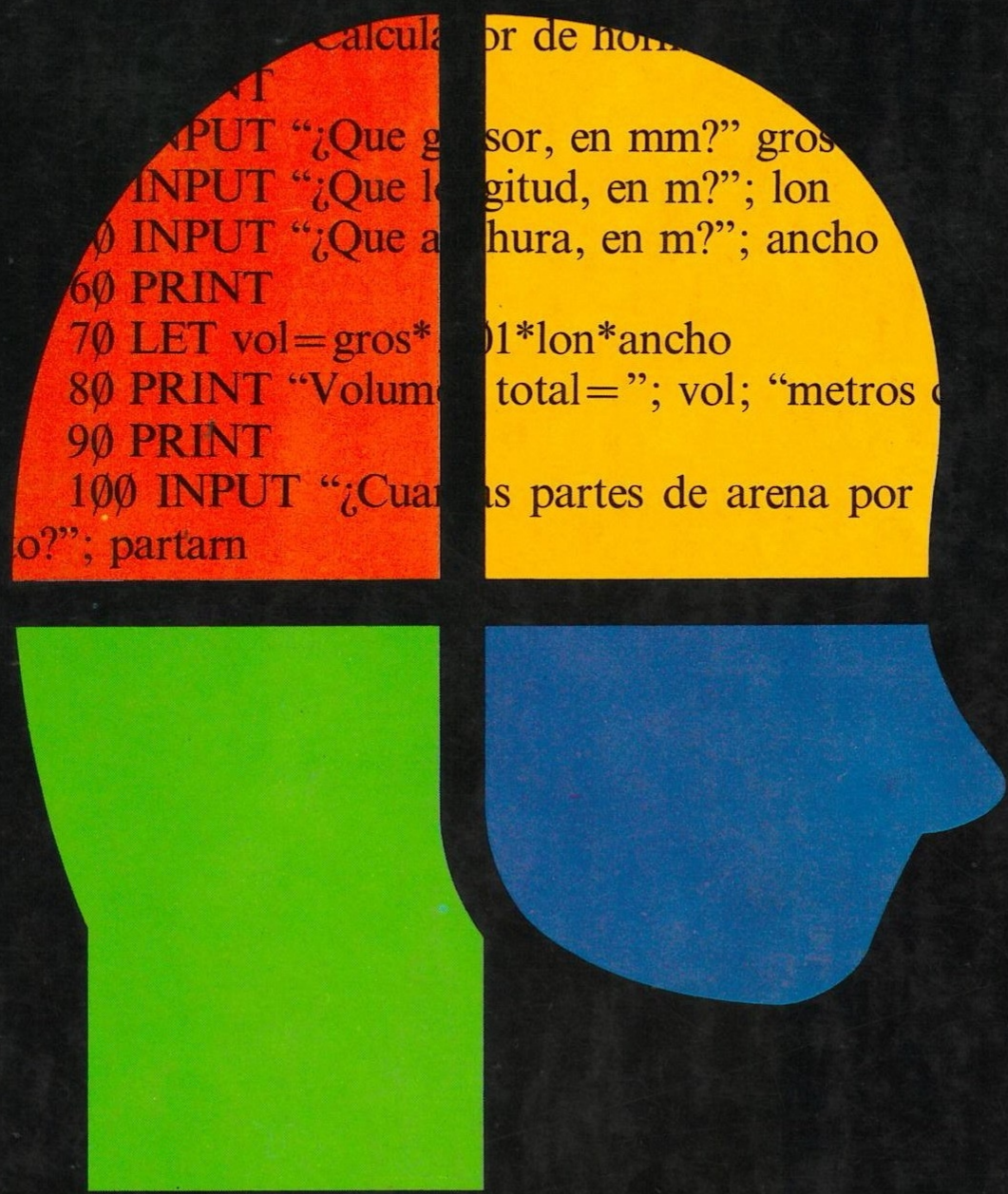
El programador de SPECTRUM



S.M.GEE

El programador de Spectrum

El programador de Spectrum

S. M. Gee

DIAZ DE SANTOS, S. A.
MADRID-ESPAÑA

Copyright © 1983 by S. M. Gee
© Granada Publishing Limited
Título original: «The Spectrum programmer»

Queda prohibida la reproducción total o parcial de la presente obra, bajo cualquiera de sus formas, gráfica o audiovisual, sin la autorización previa o escrita del editor, excepto citas en revistas, diarios o libros, siempre que se mencione la procedencia de las mismas.

ISBN: 0-246-12025-8 (edición en lengua inglesa)
ISBN: 84-86251-06-0 (edición en lengua española)

EDITA: DIAZ DE SANTOS, S. A. 1.ª edición, 1984
C/ Lagasca, 95. MADRID-6
PRODUCCION: ASEL, S. A. C/ Clara del Rey, 41. MADRID-2
Traducido por: José Ignacio Soria
Revisado por: Aula de Informática Aplicada
C/ Conde de Peñalver, 38. MADRID-6
Impreso en Lavel, Los Llanos, nave 6. Humanes (Madrid)
Depósito legal: M. 664-1985

Contenido

Prólogo

1 ANTES DE EMPEZAR 1
¿Qué es un computador? 1
Programación y programas 5
Historia del ZX-SPECTRUM 6

2 PRESENTACION DEL SPECTRUM 8
Un nuevo amigo 8
Conexión del Spectrum 9
Uso del teclado 11
Introducción de un programa 13
Más cosas acerca del teclado 16
Uso del magnetofón 20
Edición y nuestro segundo ejemplo 22
Antes de seguir adelante 25

3 LOS PRIMEROS PASOS: VARIABLES, INPUT, LET Y PRINT 26
Variables 26
Almacenando cosas en las variables: LET 28
Averiguando lo que hay en una variable: PRINT 28
Aritmética 30
Entendiendo las expresiones: Orden de los cálculos 31
Variables y constantes: Expresiones completas 32
Un programa corto 33
Otra forma de cambiar las variables: INPUT 33
Constantes y variables en expresiones 35
Descripción del BASIC 35
INPUT con comentarios 36
PRINT combinado 37
Algunos programas de muestra 38

4 SELECCION Y REPETICION: EL FLUJO DE CONTROL 42
El flujo de control 42
Bucles: Sentencia GOTO 44
Selección y condiciones: Sentencia IF 47
Empleo de IF 50
La sentencia FOR 55
Uso del bucle FOR 59
Sentencia IF con ":" 60
Ejemplo final 62

5 NUMEROS Y TEXTOS 64
Cadenas de caracteres 64
Expresiones de cadena 66
Tablas 72
Un juego de palabras 76
Inicialización de variables: DATA y RESTORE 77
Grabación de datos en cinta 79

6 FUNCIONES Y SUBRUTINAS 82
Concepto de función 82
Las funciones del Spectrum 84
Funciones definidas por el usuario: DEF FN y FN 93
Subrutinas: GOSUB y RETURN 96
Uso de las subrutinas 98

7 GRAFICOS 99
Control de PRINT 99
Funciones de impresión: TAB y AT 101
Cuando se llena la pantalla: CLS y "scroll" 104
Control de INPUT 105
Los caracteres gráficos 108
Caracteres gráficos definidos por el usuario 111
Cambio de aspecto de los caracteres: INVERSE y OVER 114
Atributos de los caracteres: FLASH y BRIGHT 117
Color: BORDER, INK y PAPER 118
Los comandos de impresión y el color 121
Colores temporales 121
Uso de los gráficos en los juegos 122

8 SONIDO Y JUEGOS 123
Sonidos sencillos: BEEP 123
Programación de melodías 125
Esperas: PAUSE 129
Algunos efectos de sonido 130
"Ataque al platillo volante": función SCREEN$ 131

9 GRAFICOS DE ALTA RESOLUCION 136
La pantalla de alta resolución 136
Los comandos gráficos: PLOT, DRAW y CIRCLE 137
Colores en alta resolución 143
Borrado de puntos: OVER e INVERSE 145
Averiguando qué hay en la pantalla: POINT 146
Uso de los gráficos de alta resolución 147

10 LOGICA Y OTROS TEMAS 150
La lógica y las expresiones condicionales 150
Dentro del Spectrum: BIN, PEEK, POKE, IN, OUT, USR y CLEAR 153
Averiguando los colores de la pantalla: ATTR 157
La sentencia REM y la buena programación 157
Después de este libro 158

Otros libros recomendados 159

Indice 161

Prólogo

Este libro ha sido escrito para todos los usuarios del Spectrum que quieran aprender a escribir sus propios programas. Programar su Spectrum puede ser muy divertido, de una gran utilidad, y, además, aprender BASIC no es en modo alguno una empresa difícil. El propósito del libro es conducirle desde los programas más elementales hasta la programación experimentada, a través de una serie de etapas claras y fáciles de seguir.

El capítulo I presenta algunas ideas que son comunes a todos los ordenadores y a la programación en general, y muestra el árbol de familia del Spectrum. El capítulo II intenta prepararle para programar su Spectrum e incluye un programa que proporcionará a éste su propia "carta de ajuste" para la televisión.

Empezaremos a aprender programación en el capítulo III, que presenta las variables y algunas de las "palabras clave" que se usan para manejarlas, tales como PRINT, INPUT, LET, etc. El capítulo IV introduce la idea de flujo de control, que es fundamental en la programación de ordenadores.

El capítulo V demuestra cómo el Spectrum maneja números y palabras y cómo usted puede combinar ambos en los programas. El empleo de las funciones del Spectrum y el concepto de subrutina asociado con ellas son los temas principales del capítulo VI, en el cual se tratan además los importantes comandos: RND, RANDOMIZE e INKEY$. El libro trata con mucha atención los gráficos y el color del Spectrum, en el capítulo VII. En el capítulo VIII, se añade el sonido y además se presenta un programa de juegos que emplea el color, los gráficos y el sonido con efectos espectaculares.

En el capítulo IX se explican los gráficos de alta resolución del Spectrum e incluye muchos programas cortos que demuestran su capacidad en este área. El último capítulo trata el tema de la lógica y presenta también algunos de los comandos que

permiten al programador controlar de manera directa el funcionamiento interno del Spectrum.

Se habrá dado cuenta ya que este libro abarca muchos temas. Sin embargo, eso no significa que vaya a ser difícil de seguir: significa que será divertido.

Más importante aún es que no se trata de un libro de lectura, sino de un libro autodidáctico. Hay montones de programas y fragmentos de programas, y todos ellos se han incluido para que usted los pruebe. Por tanto, conecte su Spectrum y disfrute aprendiendo BASIC.

Realmente, escribiendo este libro me he divertido bastante y también he aprendido algunas cosas interesantes sobre el Spectrum, que, creo, es un computador notable, y sobre BASIC, que durante mucho tiempo ha sido, en mi opinión, el mejor lenguaje de computador.

Me gustaría expresar mi agradecimiento a Jane Patience, que ayudó en la etapa menos divertida de presentar un manuscrito legible a mis editores.

Capítulo 1
Antes de empezar

Puede que cuando usted vea por primera vez el Spectrum le sorprenda su pequeño tamaño. Tal vez se sorprendería más si supiese que, hace tan sólo veinte años, un ordenador con una capacidad similar de memoria habría ocupado una vivienda familiar entera.

Usted puede preguntarse si los circuitos que hay dentro de una caja tan ligera y compacta pueden hacer, en realidad, todo lo que en los anuncios se dice que hace. Pues bien, puede hacer todas esas cosas: efectuar complicados cálculos, producir gráficos en color, emitir toda una gama de sonidos, y muchas cosas más, y cualquiera de las limitaciones que tiene no se deben realmente a su tamaño pequeño, sino que vienen impuestas por su bajo precio.

Sin embargo, el hecho de que la moderna microelectrónica haya permitido que tantas funciones puedan encerrarse en un espacio tan reducido, es algo que los usuarios del Spectrum pueden ignorar alegremente, pues al mismo tiempo que el diseño de los ordenadores se hace más sofisticado, el empleo de éstos se hace posible para todo aquel que se interese por ellos.

¿Qué es un computador?

Esta es una pregunta que puede responderse a distintos niveles. Se podrían dedicar al tema capítulos e incluso libros enteros. No obstante, para utilizar su Spectrum no necesita preguntarse acerca de este tema. Después de todo, mucha gente ve la televisión, pero muy pocos se preguntan qué es en realidad un televisor. Si usted quiere saber todos los detalles sobre la forma en que funciona el Spectrum, no le quedará otro remedio que aprender electrónica. Pero, afortunadamente, no es difícil adqui-

rir algunos conocimientos generales sobre qué es lo que hace un computador y cómo lo hace, sin saber nada de electrónica o del omnipresente "CHIP". La cuestión es que un ordenador es algo que existiría aunque jamás se hubiese inventado la electrónica. De hecho, los primeros ordenadores se construyeron con engranaje y ruedas dentadas, y tuvieron que pasar cien años antes de que la válvula (uno de los primeros componentes electrónicos) encontrara su lugar en tal máquina. Aunque en términos prácticos el ordenador parece ser un producto de la tecnología electrónica, la idea que se oculta tras él no depende de los materiales empleados para construirlo.

Todo computador consta de una serie de partes que realizan una función muy concreta. Cualquier computador debe poseer algún medio de comunicación con el mundo externo. En el caso del Spectrum, esta necesidad se satisface por medio de un teclado, en el que usted escribe, y la pantalla del televisor, que el Spectrum puede emplear para mostrarle lo que haya tecleado y cualquier otra cosa que necesite decirle. El teclado es un ejemplo de **dispositivo de entrada** y la televisión es un **dispositivo de salida.**

Estos no son los únicos dispositivos de entrada/salida (E/S) que un ordenador puede utilizar. Por ejemplo, usted puede comprar una pequeña impresora que se conecte al Spectrum y lograr que éste produzca sus salidas en la impresora en vez de, o además de, en la pantalla del televisor.

Una máquina que sólo pudiese recibir y devolver información sin cambiarla, no podría realmente considerarse como un ordenador. ¡Más bien se la podría clasificar como un teléfono o un télex!

Dentro de todo computador tiene que existir algún mecanismo que pueda cambiar o **procesar** la información antes de comunicarla al exterior. Este mecanismo toma la forma, al menos en nuestros días, de complejos circuitos electrónicos, ocultos en el interior del ordenador. El componente al que nos referimos se denomina **UNIDAD CENTRAL DE PROCESO** (UCP). En el Spectrum, la UCP está contenida en un único circuito integrado llamado Z80 y ése es, obviamente, el origen de la Z que figura en el nombre completo del aparato:ZX-SPECTRUM. Lo que hace exactamente el Z80 no tiene demasiada importancia desde el punto de vista de un programador en BASIC, y menos aún la forma en que lo hace. En términos generales, no obstante, lo que hace el Z80 es ejecutar operaciones aritméticas y otras transformaciones sobre la información

que se ha introducido desde el teclado y que está almacenada en el interior de la máquina.

Las operaciones que efectúa son controladas por una lista de instrucciones que se denomina **programa.** Este aspecto del computador es tan importante, que casi podríamos decir que un computador es una máquina que obedece una serie de instrucciones (aunque cualquier tipo de definición de una máquina tan complicada como un ordenador es arriesgada). La clase de instrucciones que el Spectrum puede obedecer ocupará el resto de este libro y, por tanto, dejaremos el tema aparte por ahora.

Si un computador tiene que obedecer una lista de instrucciones, que le indique qué debe hacer con diversos fragmentos de información, deberá tener un lugar donde almacenar no sólo la información en sí, sino además la lista de instrucciones. Esta parte del ordenador se denomina **memoria,** pero para designarla se suele emplear universalmente el nombre de **RAM** (que son las iniciales de "memoria de acceso aleatorio", en inglés). Puede usted pensar en la RAM como si fuese una especie de libreta de notas donde la UCP puede anotar su lista de instrucciones y cualquier otro dato que necesite. Naturalmente, cualquier memoria tiene una capacidad limitada y esto es una medida importante de la potencia que tiene un computador. Cuanto mayor sea la memoria, mayor será la lista de instrucciones que podrán almacenarse. La unidad más conveniente para medir la memoria del computador es el **byte.** Una memoria que puede almacenar un byte, puede almacenar, por lo general, un carácter (la palabra carácter significa aquí una letra, un dígito o un signo de puntuación de los que se encuentran en un texto corriente como, por ejemplo, en este libro). Por tanto, una memoria de 400 bytes podría almacenar suficientes caracteres como para guardar la cuarta parte de una página de este libro. El único problema con esta unidad de medida estriba en que es demasiado pequeña. Los ordenadores tienen, usualmente, memorias que pueden almacenar miles de caracteres y, por ello, será de sentido común pensar en términos de miles de caracteres. La unidad que se usa para ello es el **Kilobyte,** que a menudo se abrevia como Kbyte e incluso como K únicamente. Por diversos motivos, sin embargo, 1 Kbyte, no son 1.000 bytes como sugiere su nombre, sino 1.024 bytes (puede comprobar que este extraño número es la potencia de 2 más cercana a 1.000 y, como quizás sepa ya, los ordenadores trabajan en base binaria que se funda en el empleo de dos dígitos). Hay dos versiones del Spectrum que sólo se diferencian

en la cantidad de memoria que tienen (16K o 48K). Pero incluso el Spectrum más pequeño puede guardar aproximadamente 16.000 caracteres, lo cual es suficiente para una extensa variedad de interesantes aplicaciones (los primeros ordenadores que se usaron por los militares para calcular trayectorias de misiles, etc., a menudo tenían menos de 16K).

Esta combinación de dispositivos E/S, UCP y memoria es todo lo que constituye un ordenador. La E/S comunica al ordenador con el mundo exterior, la UCP efectúa cálculos y procesa la información, y la memoria contiene la lista de instrucciones que obedece la máquina, y los datos sobre los que actúa la UCP. En la práctica, debemos añadir una cosa a esta lista. Cuando usted desconecta el Spectrum, éste olvida todo lo que estaba almacenado en su memoria. Para mantener guardada la información con exactitud, la mayoría de las memorias de computador necesitan un suministro constante de electricidad. Si usted corta ese suministro, se perderá la información. Esta pérdida de memoria sería un grave problema, puesto que significaría tener que escribir de nuevo la lista completa de instrucciones cada vez que desenchufásemos el Spectrum. Para superar esta dificultad, la mayoría de los computadores tienen una segunda clase de memoria, que es **no volátil.** En el caso del Spectrum, está constituida por un magnetofón de cassettes normal, que puede usarse para almacenar los programas y los datos, de modo que no se pierdan, aunque desconectemos la electricidad. Una segunda ventaja de este tipo de memoria es que puede **intercambiarse.** Usted puede grabar un programa en un cassette y sacarlo después del magnetofón (e incluso enviárselo a alguien). El Spectrum está entonces listo para que usted comience un nuevo programa o regrese a uno anterior, lo cual puede hacer **cargándolo** de una cinta que estuviese grabada anteriormente. El Spectrum puede emplear también otra clase de memoria intercambiable menos común: los **MICRODRIVES.** Este dispositivo se puede comprar como accesorio del equipo básico, y se parece mucho al cassette en la forma de trabajar, sólo que es mucho más rápido. Si usted es un programador "serio", o le gustaría serlo al final de este libro, los Microdrives constituyen un deber.

Programación y programas

Como ya mencionamos, un computador obedece a una lista de instrucciones almacenada en su memoria. Esta lista de instrucciones se denomina programa, y a la escritura de tales listas se conoce con el nombre de **programación.** Frecuentemente se piensa que la programación es una actividad que comenzó con los modernos computadores digitales; pero la gente ha escrito listas de instrucciones para que las obedezcan otras personas, desde que se inventó la escritura. En ese sentido, la programación no es nada nuevo y podemos verla en las recetas de cocina y en las instrucciones para hacer punto con agujas en casi todas las casas. Tal vez uno de los mejores ejemplos de programación **tradicional** sea el de las partituras de música. Podemos ver una partitura como un programa que dirige a un músico, de forma que éste pueda interpretar una melodía. De hecho, la música escrita se parece mucho a un programa de computador, puesto que emplea un lenguaje especial que es mucho más preciso que el lenguaje ordinario. Basta que haya una sola nota fuera de su sitio, para tener una melodía diferente. Un programa de computador se escribe usando un lenguaje especial e igualmente preciso. En el caso del Spectrum, este lenguaje es BASIC, el lenguaje de programación más popular del mundo. De igual forma que en las partituras musicales, ligeros cambios en un programa BASIC pueden cambiar su significado completamente, por lo que es muy importante comprender que cuando usted aprenda BASIC debe poner atención a los más mínimos detalles desde el principio.

Al contrario que en el aprendizaje del inglés, donde usted aprende primero palabras y frases, y después los signos de puntuación, en una línea de BASIC debe saber dónde colocar cada coma, para que ésta tenga algún sentido.

Si toda esta disquisición sobre reglas estrictas le empieza a preocupar, será mejor, entonces, decir que las reglas son muy sencillas y regulares. A diferencia del inglés, en BASIC muy raras veces se encuentran excepciones a las reglas de escritura y puntuación. Además, hay toda una serie de poderosas ideas que se ocultan tras el BASIC. Una vez que usted las descubra, le harán comprender fácilmente por qué las reglas son así.

Hay dos tipos de cosas que usted aprenderá a medida que lea este libro: los detalles concretos sobre la forma exacta de cada sentencia BASIC, y las características generales que comparten

todos los lenguajes de programación. Los detalles concretos son importantes para que un programa funcione realmente; pero la comprensión de los conceptos generales hace que la programación sea un pasatiempo intelectual.

Historia del ZX-Spectrum

Antes de pasar a hablar sobre el Spectrum, sería interesante echar un vistazo a su árbol de familia. En 1980, Sinclair Research lanzó un pequeño computador con caja de plástico, el ZX-80, que puso los computadores al alcance de todos. El problema fue que era muy limitado. Fue una revolución, pero en muchos aspectos sólo estaba un poco por delante de su época. Podía usarse para ejecutar pequeños programas escritos en BASIC, pero únicamente manejaba aritmética con números enteros. Podía mostrar informaciones en una pantalla de televisión, pero sólo mientras no procesaba datos. Si estaba haciendo cualquier cosa útil, la pantalla de televisión se oscurecía. Esto supuso que mucha gente que compró el ZX-80, se decepcionase al descubrir que la máquina no respondía a lo que ellos esperaban de un computador.

En 1981 Sinclair lanzó el sucesor del ZX-80: el ZX-81.

El ZX-81 es, realmente, el primer computador útil que aparece en esta breve historia. Tenía una memoria muy pequeña (1K), pero, al menos, la pantalla no se oscurecía, y por primera vez fue posible utilizar buenos gráficos (estáticos y/o móviles). El ZX-81 es también importante, porque ha introducido el BASIC de Sinclair (o ZX-BASIC), que se usa de manera ampliada en el Spectrum. El BASIC ZX es una versión de BASIC totalmente desarrollada, que tiene muchas ventajas sobre el BASIC de otras máquinas. Para compensar la memoria tan pequeña, Sinclair produjo un módulo de 16K de RAM como accesorio, y para ampliar las posibilidades de la máquina se ofreció una pequeña impresora de bajo precio. El módulo de 16K de RAM sólo se puede usar con el ZX-81; pero la impresora aún está disponible y funciona muy bien con el Spectrum (los resultados de los programas que aparecen en este libro se produjeron usando dicha impresora).

En 1982 Sinclair anunció el Spectrum como una mejora del ZX-81. El Spectrum añadió sonido, color y gráficos de alta resolución a las habilidades del ZX-81, y ofreció unas prestaciones mejoradas, a cambio de un precio ligeramente superior. Además del Spectrum, Sinclair presentó también un dispositivo de memoria de muy bajo precio, el MICRODRIVE, y un "interface" de comunicaciones (ninguno de los cuales sirve para el ZX-81). El disponer de estos accesorios garantiza al Spectrum un lugar en las aplicaciones domésticas y profesionales durante algunos de los próximos años.

Visto como un producto formal de la gama Sinclair, el Spectrum es una máquina bien construida que reúne la experiencia adquirida con los ZX-80 y los ZX-81. Esto le ha asegurado un lugar como computador popular e importante. El BASIC ZX también ha avanzado y se ha abierto camino paralelamente. El aprender este BASIC, por tanto, le colocará en un nivel bastante bueno, ahora y en un futuro próximo.

Capítulo 2

Presentación del spectrum

Cuando se usa un ordenador, existen dos problemas: el primero es simplemente conectarlo y acostumbrarse a sus características particulares. El segundo es escribir programas que funcionen. En este capítulo estudiaremos ambos problemas de forma que queden resueltos antes de emprender nuestra principal tarea: aprender BASIC.

Un nuevo amigo

Llegar a conocer un computador es un problema que existe, incluso aunque usted sea un experto; pues aunque hay muchas cosas comunes entre ordenadores diferentes, existen siempre, también, pequeñas diferencias suficientes para que tenga que haber un período de adaptación cuando se pasa de una máquina a la siguiente.

Por ejemplo, casi todos los computadores utilizan un teclado de máquina de escribir normal (llamado QWERTY), pero la mayoría posee teclas adicionales (aunque muy importantes) en lugares ligeramente diferentes del teclado, y eso puede hacer parecer tonto, al principio, incluso al más experto. Si usted es un experto sabrá ya que esta primera fase se termina en seguida; pero si es un principiante, puede asustarse y pensar que los computadores son siempre tan complicados. Lo malo es que el no estar familiarizado con su ordenador, puede hacer que ideas sencillas sobre programación parezcan difíciles.

No hay forma de evitar esta primera dificultad para programar, ya que acostumbrarse a su computador es una cuestión de tiempo y práctica. Pasará más fácilmente por esta etapa, más bien frustrante, si tiene presentes los siguientes consejos:

(1) Distinga claramente entre cualquier dificultad que encuentre en la utilización de su Spectrum y las dificultades que tenga con la programación en sí.
(2) No suponga automáticamente que cualquier comportamiento "extraño" del Spectrum es un fallo: al principio, lo más probable es que la equivocación sea suya.
(3) Tampoco piense en seguida que cualquier conducta inesperada de un programa significa que su Spectrum es ilógico. Los computadores son inhumanamente lógicos.
(4) Intente no confundir los errores de escritura con los errores de programación.

Para ayudarle a que identifique esta dificultad inicial y pueda superarla, este capítulo incluye dos programas cortos que debería intentar hacer funcionar en su Spectrum antes de continuar con el resto del libro. Ambos programas se presentan completos, y correctos para que los emplee en descubrir los problemas que surgen al trabajar con el Spectrum y que no son de programación.

En esta etapa no se pretende que pueda entender cómo funcionan, y tal vez le gustaría volver a ellos cuando lea posteriores capítulos, para comprobar cómo progresa.

Conexión del Spectrum

El Spectrum es uno de los computadores más sencillos del mundo de poner en marcha. Todo lo que tiene que hacer es enchufar la fuente de alimentación (la pequeña caja negra que se llama ZX POWER SUPPLY) a la red, e insertar la pequeña clavija de ésta en el enchufe de la parte trasera del Spectrum, marcado con 9V DC. En ese momento debería escuchar un agudo zumbido, que suena como un mosquito insistente, que procede del Spectrum. Si no escucha nada, compruebe que el enchufe de la red que está empleando funciona correctamente. Si es así y aún no puede oír nada, entonces pulse la tecla marcada con "A" a la izquierda del teclado. Si mantiene pulsada la tecla, debería escuchar un "click" que viene del computador. Esta prueba no debería fallar con ningún Spectrum que funcione.

La conexión a la red es la única cosa que se requiere para hacerlo funcionar. Sin embargo, si usted quiere ver lo que está haciendo tendrá que conectarlo a un aparato de televisión con

UHF. Esta operación es un poco más difícil que conectarlo a la red, puesto que comprende un aparato que variará mucho de una casa a otra: el televisor. Cuando conecte el Spectrum al televisor, le ayudará pensar en él como si fuese un canal más. Sí, el Spectrum no es únicamente un computador: ¡es además una pequeña emisora de televisión! Al comprar su televisor, éste venía sintonizado para captar los canales que se reciben en su zona. De igual forma, cuando utilice el Spectrum por primera vez, tendrá usted que sintonizar el televisor para que lo reciba. Si su televisión tiene botones para sintonía preseleccionada, entonces tendrá que decidir cuál de los botones va a ser el "canal del Spectrum", y averiguar en el manual de instrucciones del televisor cómo sintonizarlo. Si el televisor tiene un mando de sintonía continua, entonces encontrar el canal del Spectrum es igual que encontrar cualquier otro canal. A pesar de ello, antes de que enchufe el Spectrum al aparato de televisión, sintonice el canal 36 de UHF usando el método apropiado en su televisor.

Una vez sintonizado en el canal 36, tome el cable que viene con el computador (con una clavija aérea de televisión en un extremo) e insértelo en la toma aérea del televisor (¡sin olvidar desconectar antes la clavija del cable original, claro!). Después conecte el otro extremo a la toma trasera del Spectrum marcada T.V.

Ahora, con el Spectrum y el televisor encendidos (debe bajar el volumen de la televisión para evitar sonidos poco agradables) empiece a mover cuidadosamente el mando de sintonía del televisor. Cuando se acerque al canal exacto del Spectrum, empezará a ver una imagen difusa. Continúe haciendo lo mismo hasta que el mensaje:

© 1982 Sinclair Research Ltd.

se pueda observar claramente en la parte inferior de la pantalla. Como en cualquier canal de televisión, si no lo sintoniza correctamente, la calidad de la imagen será más bien pobre; por lo tanto, hágalo con cuidado y tenga paciencia con el mando de sintonía hasta que obtenga una imagen clara y definida.

Si está usando un aparato en color, tal vez tenga que ajustar el brillo, el contraste y el color hasta obtener una buena imagen; pero no haga esto hasta que termine de sintonizar correctamente y logre una imagen lo más definida posible, o las cosas se volverán bastante confusas. Si su televisor es en blanco y negro,

no podrá ver los colores que produce el Spectrum, pero quizás tenga también que ajustar los controles de brillo y contraste del televisor para obtener una imagen satisfactoria. Un televisor en blanco y negro funcionará perfectamente con el Spectrum, sólo que en lugar de ver diferentes colores, podrá ver ocho tonalidades diferentes de gris comprendidos en la gama del negro al blanco.

Si quiere, puede dejar los ajustes finales hasta que haya introducido el programa que daremos más tarde en este capítulo, puesto que éste producirá una carta de ajuste propia del Spectrum, que le ayudará a lograr una sintonización perfecta.

Uso del teclado

Después de poner en marcha el Spectrum, lo siguiente que hay que hacer es "vérselas" con el teclado. Quizás la característica más incómoda del Spectrum sea la cantidad de letras y palabras que están impresas encima, en la propia tecla, y bajo la misma, en todo el teclado.

La idea de que cada tecla lleve a cabo más de una función no es nueva. La mayoría de las máquinas de escribir utilizan una sola tecla para imprimir las letras mayúsculas y minúsculas, y nadie parece transtornarse por tener que elegir entre ambos tipos pulsando una tecla extra (la tecla de cambio: SHIFT). Observe que la tecla SHIFT no escribe nada si se pulsa sola, por lo que no es igual que las demás teclas; puesto que sirve para controlar lo que escribirán otras teclas, se denomina **tecla de control.**

El Spectrum tiene toda una serie de teclas de control que se usan en diferentes combinaciones para acceder a todos sus caracteres, comandos y otras cosas. El diagrama (Figura 2.1) muestra su situación en el teclado. La función de todas estas teclas de control se explicará en este capítulo, empezando con las que se necesitan con mayor frecuencia.

El Spectrum emplea dos teclas de control para seleccionar entre tres conjuntos de caracteres distintos impresos en cada tecla. La tecla CAPS SHIFT funciona de la misma manera que la tecla de cambio de una máquina de escribir. Si usted pulsa una tecla, logrará la versión minúscula de la letra escrita sobre ella. Sin embargo, si la pulsa mientras mantiene oprimida la tecla CAPS SHIFT, obtendrá la letra mayúscula impresa en la tecla.

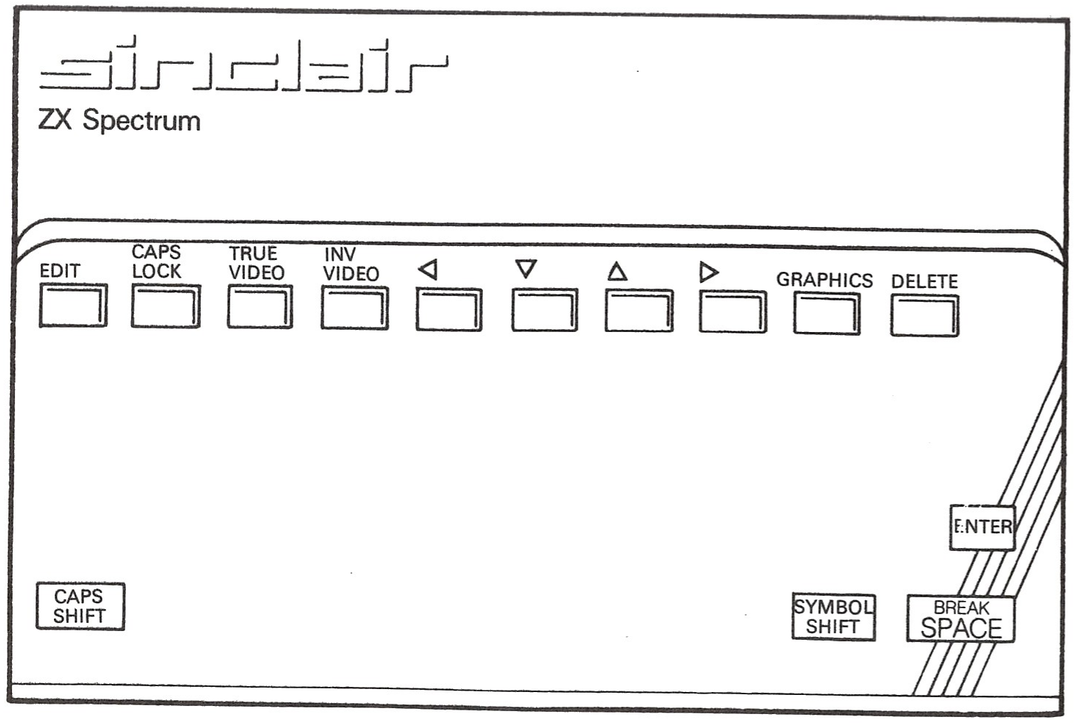


Fig. 2.1. Posición de las teclas de control.

La segunda tecla de control se llama SYMBOL SHIFT (cambio de símbolo) y está en la parte inferior derecha del teclado. Pulsando cualquier tecla mientras se mantiene pulsada esta última, se produce la palabra o símbolo escritos en color rojo a la derecha de la letra grande impresa en la tecla.

El manejo de las dos teclas Shift parece bastante sencillo; pero si acaba de encender el Spectrum y oprime la tecla "A", se sorprenderá al comprobar que en la pantalla aparece la palabra NEW: ¡algo ha ido mal, evidentemente! Con las reglas de antes, al pulsar la tecla "A" debería aparecer una "a" minúscula. La razón de esta discrepancia es que cualquier línea de BASIC empieza con una palabra tomada de las de un pequeño conjunto de palabras, llamadas **palabras clave.** Para facilitar las cosas, el Spectrum interpreta la primera tecla literal que se pulsa como una palabra clave. La palabra clave que escribirá cada tecla está escrita en blanco en la parte inferior de ésta. Por ejemplo, la palabra clave asociada a la tecla "A" es NEW y por eso aparece en la pantalla cuando se pulsa la tecla "A" por vez primera. Si pulsa la tecla "A" una segunda vez, verá aparecer la letra "a" en la pantalla, como habíamos dicho antes. Para que sepa lo que una tecla va a escribir cuando la pulse, el Spectrum imprime

diferentes letras en el interior del cuadrado intermitente que hay en la parte baja de la pantalla. Este se llama **cursor.** Su posición en la pantalla indica dónde se escribirá lo siguiente que introduzca, y la letra de su interior informa sobre qué conjunto de caracteres se utilizará. Al principio de cada línea, el cursor consiste en una [K] intermitente, hasta que se introduce una palabra clave y después cambia a una [L] intermitente ([K] viene de "Keyword", en inglés, palabra clave y [L] de "Letter", letra en inglés). Mientras el cursor sea una [L] intermitente, las dos teclas "Shift" funcionarán como dijimos al principio.

Para resumir, observemos una tecla determinada como la mostrada en la figura:

Las palabras claves escritas en blanco en la parte inferior de las teclas se obtienen cuando el cursor es una [K] intermitente. En nuestro ejemplo, obtendríamos un "RUN".

Las letras mayúsculas y minúsculas se obtienen cuando el cursor es una [L] intermitente, las minúsculas cuando CAPS SHIFT no está pulsada (lo que corresponde a una "r" en este caso) y las mayúsculas ("R", aquí) cuando está pulsada CAPS SHIFT al mismo tiempo que la letra en cuestión.

Los símbolos o palabras de color rojo, escritos en la parte superior derecha de las teclas, se obtendrán cuando se pulse la tecla al mismo tiempo que SYMBOL SHIFT. En este caso, lograríamos un "<" (símbolo menor). Esto último se aplica cuando el cursor muestra tanto una [K] como una [L].

Ahora sabemos cómo obtener 4 de los símbolos o palabras que rodean cada tecla, y esto es suficiente para introducir un programa corto. Hay más cosas sobre el uso del teclado, pero volveremos a ello posteriormente.

Introducción de un programa

Como se explicó en el primer capítulo, un programa es una lista de instrucciones que el computador puede obedecer. En BASIC, esta lista está constituida por una serie de líneas de

comandos, que comienzan todas con un número llamado **número de línea.**

La forma más simple de entender todo esto es probarlo con un programa corto.

No se preocupe si no comprende cómo funciona el programa. Concéntrense únicamente en introducirlo correctamente. Antes de empezar, apague el Spectrum, y, tras esperar un instante, vuelva a encenderlo. Esto producirá el familiar mensaje de Sinclair que vimos antes en la pantalla, y asegura que cualquier cosa que pueda haber escrito antes, mientras experimentaba, se ha borrado de la máquina. Ahora escriba la línea que sigue, cuidando de no equivocarse (si se equivoca, vuelva a desenchufar y enchufar una vez más. Esto sólo es una forma temporal de corregir los errores; dentro de poco veremos métodos mejores).

```
1Ø LET a=Ø
```

Observe que LET es una palabra clave, y se obtiene pulsando la tecla "L" mientras el cursor muestra una [K] intermitente. Dese cuenta también de que el segundo y el último carácter son ceros, no la letra "O" (el cero—Ø— está en el extremo derecho de la fila superior del teclado). El signo "=" está en rojo en la tecla "L" y, por tanto, para obtenerlo debe pulsar SYMBOL SHIFT y "L".

Después de que haya escrito esta línea, tiene que pulsar la tecla ENTER del extremo derecho del teclado. El propósito de esto es decirle al Spectrum que ha terminado usted de escribir la línea y que él puede "intentar" añadirla a cualquier trozo de programa que ya hubiese escrito. Decimos "intentar", pues aunque usted crea que lo que ha escrito es correcto, el Spectrum lo comprueba, y si encuentra que ha escrito algo sin sentido, lo rechazará. Cuando el Spectrum acepta la línea, ésta desaparece de la parte inferior de la pantalla y aparece en la parte superior. Ahora introduzca la línea:

```
2Ø PRINT a
```

De nuevo, la palabra PRINT es una palabra clave y se introduce pulsando una sola tecla, en este caso la "P". Pulse ENTER y la segunda línea reaparecerá de nuevo en la parte superior de la pantalla, inmediatamente debajo de la primera línea. Vamos esta vez con la línea:

```
3Ø LETa=a+1
```


Esta vez, antes de que pulse ENTER, veamos cómo podría usted corregir cualquier error que pudiese haber cometido. Suponga, por ejemplo, que por equivocación ha escrito b=a+1. Le alegrará saber que el Spectrum le ofrece una tecla de retroceso y le permite cambiar fácilmente la línea incorrecta que haya escrito. Si pulsa la **flecha hacia la izquierda** (el 5 de la línea superior del teclado) mientras está oprimiendo CAPS SHIFT, verá que el cursor se mueve a través de las letras que había escrito. Si oprime la tecla DELETE (el Ø de la fila superior) mientras pulsa CAPS SHIFT, la letra, e incluso toda la palabra clave que esté a la izquierda del cursor, desaparecerá (es decir, será borrada de la línea). Puede insertar una nueva letra o palabra clave, simplemente pulsando la tecla correspondiente. Puede mover el cursor a la derecha pulsando la **flecha hacia la derecha** (la tecla "8" de la fila superior) y cualquier nuevo carácter que usted introduzca, se insertará inmediatamente después de la posición del cursor (todo esto es mucho más fácil de ver haciéndolo que leyendo la explicación. No dude en experimentarlo todo, no puede estropear con ello su Spectrum).

Finalmente, cuando haya terminado de escribir la línea correcta, pulse ENTER.

Ahora, teclee:

```
4Ø GOTO 2Ø
```

Compruebe que, cuando pulsa la tecla etiquetada con GOTO, aparecen en la pantalla ambas palabras GO TO.

En este momento, debería tener en la parte superior de la pantalla el siguiente programa:

```
1Ø LET a=Ø
2Ø PRINT a
3Ø LET a=a+1
4Ø GOTO 2Ø
```

Esta es una lista de instrucciones, y usted puede hacer que el Spectrum obedezca introduciendo la palabra clave RUN y pulsando después ENTER. Pero antes de que lo intente, es mejor informarle de que hay dos formas en que el Spectrum puede obedecer comandos. Si escribe usted un comando sin número de línea y pulsa ENTER después, el Spectrum ejecutará inmediatamente ese comando; esto se llama **modo inmediato.** Sin embargo,

si antepone un número de línea al comando, y pulsa ENTER, éste es incorporado a la parte de programa que ya estuviese introducida y este programa permanecerá almacenado para ejecutarlo más tarde, cuando esté completo; eso es el llamado **modo programa.**

Cuando escribe RUN, no hay ningún número de línea que preceda a esta orden, y por ello el Spectrum la ejecuta inmediatamente.

En el momento en que escriba RUN en el programa que ya ha introducido, podrá ver que se imprimen en la pantalla los números del Ø a 21 y después el Spectrum imprime "algo" que parece ser una pregunta: "Scroll? y/n". Si pulsa la tecla "Y", observará que la pantalla se desplaza una línea hacia arriba para dejar sitio libre y poder imprimir más información. Una vez que esto ha ocurrido 22 veces, otra vez aparece la misma pregunta. El motivo de esto es que el Spectrum intenta darle la oportunidad de ver tranquilamente lo que hay en la pantalla, antes de desplazarla hacia arriba para imprimir nuevas cosas.

Podrá seguir respondiendo con "Y" a esta pregunta de "Scroll?", hasta que se canse de ver aparecer números en la pantalla. Cuando eso suceda, responda "N" (por aquello de "NO") en la siguiente ocasión en que aparezca la pregunta, y después de esto pulse ENTER. Verá aparecer de nuevo en la pantalla un listado del programa.

Si mientras el programa está funcionando, quiere detenerlo en cualquier momento, pulse la tecla marcada con BREAK en tanto que mantiene oprimida la tecla CAPS SHIFT. Como otro ejemplo de un comando en modo inmediato, teclee LIST (la tecla marcada "K") y pulse ENTER. Eso hará que aparezca en la pantalla el programa que usted escribió.

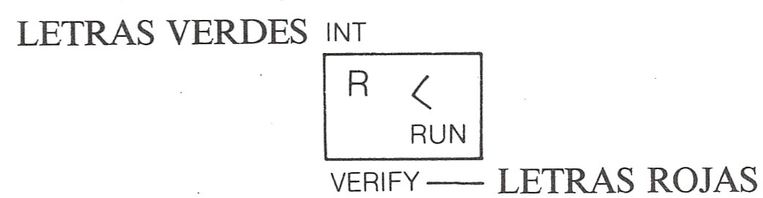
Más cosas acerca del teclado

Para introducir programas de mayor complejidad que el anterior, hay algunos símbolos y palabras que necesitaremos saber cómo obtener. En particular, aún no sabemos cómo obtener las palabras o símbolos que están impresos en la caja del Spectrum encima y debajo de cada tecla. Realmente, todos esos nuevos símbolos y palabras se introducen cuando el cursor muestra un nuevo modo. Al igual que las palabras claves se obtienen cuando el cursor es de la forma [K] intermitente, los

símbolos rojos y verdes y las palabras de igual color mencionadas se obtienen en el llamado **modo extendido,** que se caracteriza porque el cursor aparece como una [E] intermitente. Puede poner al Spectrum en este nuevo modo de trabajo, pulsando juntas las teclas de Shift (SYMBOL SHIFT y CAPS SHIFT). Las palabras en verde que están encima de cada tecla se obtienen presionando la tecla correspondiente en modo extendido, y las palabras en rojo debajo de cada tecla se obtienen presionando ésta junto con SYMBOL SHIFT.

Eso es fácil de recordar, puesto que para obtener cualquier símbolo o palabra en **rojo** tiene que pulsar la tecla SYMBOL SHIFT, que tiene las letras también de ese color. De esta forma, para introducir RND (encima de la tecla "T") primero tendrá que pasar al modo extendido, usando CAPS SHIFT y SYMBOL SHIFT a la vez, y después pulsar "T". Si quiere obtener BEEP (debajo de la tecla "Z"), primero pase a modo extendido y pulse después SYMBOL SHIFT y "Z" al mismo tiempo. El modo extendido se abandona automáticamente, tan pronto como escriba algo; pero si quiere salir de él sin escribir nada, puede pulsar de nuevo las teclas Shift.

Para hacer un resumen de esta nueva información, ampliemos el ejemplo que teníamos antes. La figura siguiente muestra la tecla "R" y todos los rótulos que la rodean y que puede lograr escribir:



Las palabras de color verde que hay sobre cada tecla se obtienen cuando el cursor es una [E] intermitente, después de haber pulsado a la vez SYMBOL SHIFT y CAPS SHIFT. Una vez que haya aparecido la [E], si pulsa la tecla "R" observará INT en la pantalla.

Las palabras de color rojo escritas debajo de cada tecla, se obtienen cuando el cursor es una [E] intermitente y pulsamos la tecla correspondiente junto con SYMBOL SHIFT (o junto con CAPS SHIFT). En este caso, con esas acciones lograríamos VERIFY.

Ahora ya sabe cómo obtener casi todos los símbolos y pala-

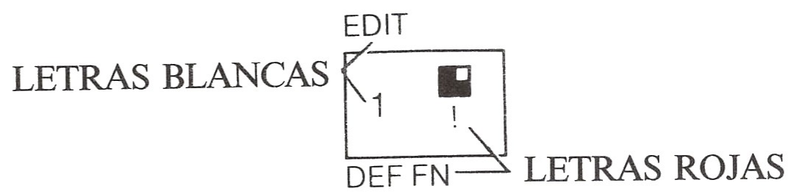
bras del teclado. Lo único que queda por ver para que no tenga ningún problema son los caracteres que están en las teclas de la fila superior del teclado. Estos caracteres deben introducirse en otro modo: el **modo gráfico.** Aunque trataremos con todo detalle el tema de los gráficos en el capítulo VII, tal vez podemos decir aquí cómo obtener esos caracteres y cómo pasar al modo gráfico. Para pasar a éste, todo lo que tiene que hacer es pulsar GRAPHICS (que está impreso encima de la tecla 9, en la fila superior) mientras al mismo tiempo, presiona CAPS SHIFT. Eso hace que el cursor cambie a una [G] intermitente, y si ahora pulsa cualquier tecla de la fila superior, salvo 9 y Ø, verá cómo se imprimen en pantalla los caracteres gráficos que se encuentran pintados en las teclas de esa fila en color blanco. Si pulsa las teclas anteriores, pero esta vez al tiempo que mantiene pulsada CAPS SHIFT, comprobará que en la pantalla se imprime ahora el carácter correspondiente, pero invertido, es decir, lo que era blanco ahora es negro y viceversa.

A lo mejor se está preguntando qué ocurriría si pulsase las teclas de la fila superior (las de los números) a la vez que CAPS SHIFT cuando estamos en modo [L]. ¡Al fin y al cabo, los números no tienen mayúsculas ni minúsculas! La respuesta es que no se producirán nuevos caracteres. En lugar de eso, tendrá usted acceso a las restantes funciones de control que aparecen en la Figura 2.1. Utilizando la tecla "1" junto con CAPS SHIFT, veremos cómo la línea del programa (en la parte superior de la pantalla) señalada por un ">" (el cursor del programa), baja a la parte inferior de la pantalla para que pueda **editarla.** Ya veremos, más avanzado este capítulo, para qué vale esto. Si pulsa la tecla "2" con CAPS SHIFT, activará la función CAPS LOCK. La primera vez que lo haga, verá como ahora el cursor es una [C], cuando aparentemente debiera ser una [L], y, además, todas las letras que pulse se escribirán en mayúscula. Para volver a tener las minúsculas tienen que pulsar de nuevo CAPS SHIFT y "2" a la vez. También verá que encima de la tecla "4" pone INV. VIDEO. Otra vez tendrá que pulsar CAPS SHIFT y "4" juntas, para tener acceso a esta función, cuyo efecto es escribir invertidos en la pantalla los caracteres que introduzca después de activarla: aparecen escritos en blanco sobre fondo negro, en vez de negro sobre blanco. Si quiere volver a tener todo como siempre, pulse CAPS SHIFT y "3", que repondrá el modo TRUE VIDEO, como indica el rótulo sobre la tecla "3".

Pulsando CAPS SHIFT a la vez que las teclas "5", "6", "7" u "8" podrá mover el cursor por la pantalla. Estas teclas se usan tanto en la edición de textos como en los juegos. Ya vimos como funcionaban las "flechas" escritas sobre "5" (la izquierda) y sobre "8" (la derecha), y las otras dos se explicarán muy pronto.

Hemos dicho antes que con CAPS SHIFT y "9" a la vez obteníamos el modo gráfico señalado por el cursor [G] intermitente. Si está ya en ese modo, restaurará el modo [L] original volviendo a pulsar la misma combinación: CAPS SHIFT y Ø, a la vez, le permiten borrar lo que haya a la izquierda del cursor, cuando esté escribiendo o corrigiendo una línea.

Las teclas numéricas son bastantes diferentes a las demás del Spectrum, y es preferible que las examinemos en detalle antes de dejar el asunto del teclado por un momento. El diagrama que sigue muestra la tecla que está más a la izquierda en la fila superior, la tecla "1", y lo que sigue será un resumen de cómo producir todos los símbolos y palabras relacionados con ella (lo mismo valdría para las demás teclas numéricas):



Cuando tengamos una [K] o [L] intermitentes, al pulsar una de las teclas numéricas obtendremos el propio número que está escrito en ella, el "1" en este caso. Si se pulsa la tecla a la vez que SYMBOL SHIFT, se produce el símbolo rojo escrito en la parte inferior derecha, que aquí corresponde a la admiración.

Si se pulsa la tecla a la vez que CAPS SHIFT, estaremos utilizando la función de control indicada encima de la tecla. Ya hemos detallado para qué sirven las funciones de control de cada tecla numérica. Para el "1" correspondería a EDIT, que sirve para bajar una línea de programa a la parte inferior de la pantalla y prepararla para editarla (corregirla o cambiarla de algún modo).

Cuando estemos en modo [E], extendido, y pulsemos la tecla al tiempo que SYMBOL SHIFT, produciremos el comando escrito en color rojo sobre la caja del Spectrum, debajo de la tecla. En el ejemplo sería DEF FN, que sirve para definir funciones.

El símbolo gráfico de la tecla se puede obtener pulsando ésta en modo gráfico, y para el "1" puede ver que sería un cuadrado negro al que le falta la esquina superior derecha.

Una propiedad del teclado que aún no hemos comentado, es la capacidad que tiene de repetir una tecla cuando ésta se mantiene un cierto tiempo. Si deja el dedo sobre la tecla cuando está el cursor □, verá como la línea se rellena con el carácter correspondiente, y podrá oír cómo se produce un sonido "click" repetido en su Spectrum.

En este momento, podría pensar que el teclado del Spectrum es la cosa más complicada con la que se haya enfrentado nunca. Y, en verdad, es complicado; pero cuando lo use repetidas veces verá que su funcionamiento es bastante lógico.

Uso del magnetofón

El siguiente programa de demostración es bastante más largo que el primero, y seguramente le desagradará apagar la máquina y perderlo entero después de haberlo introducido y ejecutado y, por ello, ahora es el momento de aprender el uso de un magnetofón de cassettes para guardar y recuperar programas.

El Spectrum puede usar casi cualquier magnetofón para almacenar programas; pero lo cierto es que cuanto mejor sea éste, más fiable será el resultado. De manera que, si estaba pensando en comprar un cassette para usarlo con su flamante Spectrum, invierta como mínimo 5.000 pesetas, e intente comprar un modelo que tenga tomas de auriculares (EAR) y micrófono (MIC) para "jacks" miniatura (de 3,5 mm; el jack es una clavija). Si el aparato que quiere usar no tiene tales conexiones, tendrá que comprar un adaptador en una tienda de sonido, pues ésas son las tomas con las que viene el computador.

Para conectar el Spectrum al magnetofón, sólo tiene que conectar las dos clavijas del cable de audio, que viene con el ordenador, en las dos tomas del Spectrum, que están en la parte trasera marcadas MIC y EAR (el cable de audio lo reconocerá fácilmente, pues es el único que no ha usado hasta ahora). No importa en este momento qué clavija conecta a cada toma. Después, enchufe el cable del mismo color que el que conectó a la toma MIC del Spectrum en la toma MIC de su grabador. Ya está todo preparado para que grabe su primer programa.

Para tener en el Spectrum algo que poder grabar, teclee el programa:

```
10 REM esto es una prueba
20 REM esta es la segunda linea
30 REM esta es la tercera linea
```

y después escriba:

```
SAVE "prueba"
```

y pulse ENTER. Aparecerá el mensaje: "start tape, then press any key" (ponga en marcha la cinta, después pulse cualquier texto). Quiere decir que el ordenador está listo para grabar el programa en la cinta y está esperando a que usted le diga que la cinta está en marcha. Inserte una cinta en blanco, coloque el nivel de grabación aproximadamente a la mitad de su recorrido (esto no es necesario si el magnetofón tiene nivel automático de grabación) y encienda el magnetofón en posición de grabación. Pulse cualquier tecla del Spectrum y verá unas líneas horizontales, enmarcando la pantalla, que significan que el programa se está grabando. Cuando aparezca el mensaje "OK" en la pantalla, querrá decir que el programa se ha grabado en la cinta. Sin embargo, podría ocurrir que, aunque esté grabado en la cinta, la calidad de la grabación no fuese suficientemente buena como para leerlo después desde el computador, lo cual podría ocurrir por diversos motivos.

Para averiguar si el programa se ha grabado correctamente, rebobine la cinta, **desenchufe** la clavija de la toma MIC del grabador y **conecte el otro cable** en la toma EAR (de auriculares) de éste. Por fin escriba:

```
VERIFY "prueba"
```

y pulse ENTER. Encienda el cassette, y si todo ha ido bien volverá a aparecer en pantalla el mensaje "OK", que indica que lo que se ha leído de la cinta es igual que lo que el computador tiene en memoria. Si algo falló, el mensaje que aparece entonces es: "R Tape loading error", o simplemente nada. Eso indicaría que, por algún motivo, el Spectrum ha leído mal la información de la cinta e incluso que no ha podido encontrarla. En estos casos, lo mejor es rebobinar una vez más y quitar el enchufe de

la toma EAR del grabador, para poder "oír" lo que ha ocurrido. Al principio de la cinta debería escuchar una especie de "pitido" alto, pero no distorsionado. Si lo escucha, avance la cinta y, con el mismo volumen, compruebe si se oye en un trozo sin grabar algo como un silbido apagado. Si lo que se escucha parece una "tormenta", baje el volumen y mueva el control de tono e insértelo de nuevo. Si en un trozo de cinta sin grabar no puede oír nada, debería subir el volumen y repetir la operación.

Después de usar VERIFY para comprobar si el programa se ha grabado correctamente, puede intentar ahora cargarlo realmente. Primero desenchufe el Spectrum y vuelva a enchufarlo para convencerse de que el programa de prueba se ha borrado, y después teclee:

LOAD "prueba"

o

LOAD""

y pulse ENTER. La primera versión del comando LOAD hará que se busque en la cinta hasta encontrar un programa con el nombre de "prueba"; pero la segunda versión cargará el primer programa de la cinta, sea cual sea su nombre.

Una vez que logre hacer funcionar el sistema de cinta, no mueva los controles de tono y volumen, a menos que sea necesario y **recuerde** cambiar las clavijas cada vez: conecte **sólo** la toma MIC para grabar programas, y sólo la toma EAR para cargarlos de la cinta. Esto es muy importante, pues en otro caso el sistema no funcionará de manera fiable.

Edición y nuestro segundo ejemplo

A estas alturas, debería sentirse suficientemente confiado como para atreverse a introducir un programa más largo. El programa que viene a continuación producirá una "carta de ajuste" como la que aparece en la Figura 2.2.

Además de ser un gráfico muy interesante, puede usarlo para ajustar el televisor que esté utilizando con su Spectrum y producir así una imagen lo más definida posible y con el mejor color.

Antes de empezar a introducir el programa, los siguientes consejos podrían serle útiles:

Teclee primero NEW para borrar cualquier programa anterior.

Todas las palabras, salvo "T.V. SPECTRUM", en la línea 13Ø, y "CANAL 36", en la 14Ø, se introducen usando una única tecla; por lo tanto, busque por el teclado hasta que encuentre la palabra adecuada. Observe que en la línea 11Ø, el signo menos que precede al número 175 se obtiene pulsando la tecla "J" a la vez que SYMBOL SHIFT.

Debido a la dificultad para imprimir los caracteres que se introducen en modo gráfico, usamos una notación especial. Siempre que vea un carácter entre corchetes, significará que dicho carácter debe introducirse en modo gráfico. Si además va precedido por una fleha, "↑", deberá pulsar CAPS SHIFT a la vez que el carácter, y siempre en modo gráfico. Por ejemplo: "[↑8]" en las líneas 17Ø, 18Ø, 19Ø y 2ØØ, significa: "poner el modo gráfico y luego pulsar 8 a la vez que CAPS SHIFT". El resultado, en este caso, debe ser un cuadrado negro.

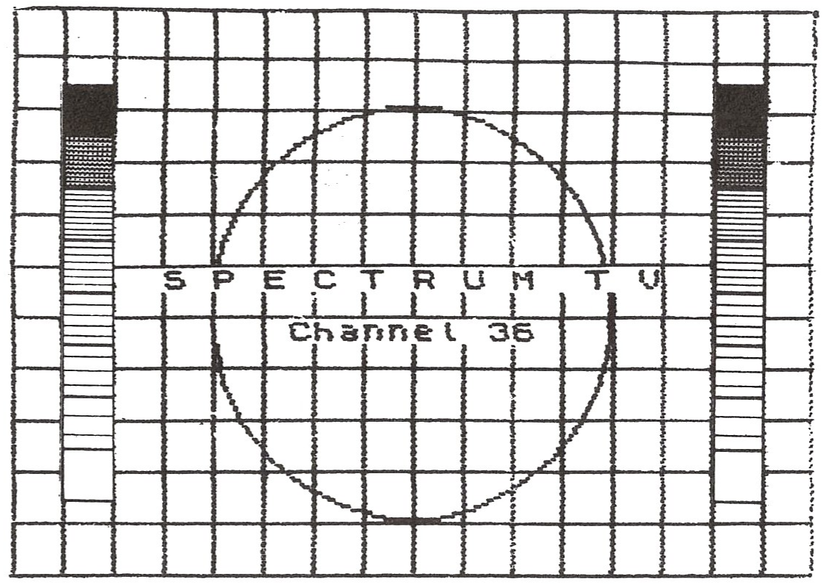


Fig. 2.2. Carta de ajuste del Spectrum.

Cuando haya terminado de introducir el programa, pulse RUN y verá aparecer la carta de ajuste en la pantalla. Si ésta no es igual que la Figura 2.2, probablemente habrá cometido algún error al teclear y debería revisar cuidadosamente el programa con el listado del libro. Puede que se le haya olvidado alguna línea entera. Ese problema es fácil de solucionar: para insertar

```
10 FOR i=0 TO 240 STEP 16
20 PLOT i,0
30 DRAW 0,175
40 NEXT i
50 FOR i=0 TO 175 STEP 16
60 PLOT 0,i
70 DRAW 255,0
80 NEXT i
90 PLOT 0,175
100 DRAW 255,0
110 DRAW 0,-175
120 CIRCLE 127,81,64
130 PRINT AT 10,6; "T V  S P E C T R U M"
140 PRINT AT 12,11; "Channel 36"
150 FOR c=0 TO 7
160 INK c
170 PRINT AT 3+2*c,2;"[↑8]"
180 PRINT AT 4+2*c,2;"[↑8]"
190 PRINT AT 3+2*c,28;"[↑8]"
200 PRINT AT 4+2*c,28;"[↑8]"
210 NEXT c
220 INK 0
```

una nueva línea sólo tiene que escribirla, sin olvidar su número al principio, y aparecerá en la parte inferior de la pantalla.

Presione ENTER, y la línea se colocará automáticamente en el sitio exacto del programa que le corresponda según su número. Normalmente, los programas tienen las líneas numeradas en intervalos de diez en diez para facilitarle la inserción de alguna nueva línea que pudiese necesitar más adelante. Si quiere borrar una línea entera de un programa, escriba únicamente su número, pulse ENTER, y la línea desaparecerá en el acto (por ello, tenga cuidado de no borrar ninguna línea de las que quiera conservar.

Si observa alguna línea errónea, no tiene que volver a escribirla entera para corregirla; en lugar de ello, puede **editarla.** Si usa las teclas con flecha hacia arriba y hacia abajo (7 más CAPS SHIFT y 6 más CAPS SHIFT, respectivamente), mientras el programa aparece listado en la pantalla, notará que puede mover el cursor hacia arriba y hacia abajo de la pantalla para que señale la línea que usted desee. Entonces, si lleva el cursor a la línea que tenía el error que usted, hipotéticamente, descubrió, y pulsa EDIT (tecla 1 más CAPS SHIFT), la línea bajará a la

parte inferior de la pantalla (el **área de entrada)** donde estaba la primera vez que la escribió. Una vez ahí, puede usar las teclas con las flechas hacia la izquierda y hacia la derecha (escritas sobre 5 y 8) y la tecla DELETE (borrar: 0 más CAPS SHIFT) para corregir cualquier equivocación existente (a esto se denomina **edición).** Cuando termine de editarla, pulse ENTER y automáticamente la línea, ya corregida, se colocará una vez más en su lugar correspondiente dentro del programa.

Algunas veces puede ahorrar tiempo, cuando esté escribiendo un programa, usando esta facilidad de edición. Por ejemplo, las líneas desde 170 a 200 son muy similares. En vez de introducirlas de una en una, puede introducir la línea 170 y después (con el cursor señalándola) pulsar EDIT. Así copiará ésta en el área de entrada donde puede cambiar el número de línea a 180 y el "3" que va detrás de AT, por un "4". Si pulsa ENTER, la línea 180 aparecerá en el programa, sin haber tenido que escribirla entera.

Antes de seguir adelante

A medida que aprenda BASIC y las características especiales del Spectrum, leyendo el resto de este libro, irá mejorando tanto en el uso del teclado como en su comprensión de la programación y de los programas, de forma que, en un momento dado, no se explique cómo le parecía todo tan complicado al principio. Sin embargo, hasta que eso ocurra, es bueno que recuerde el consejo que le dimos al principio de este capítulo: no permita que la frustración que provocan los errores de tecleo interfiera en su comprensión de la programación en general y del BASIC en particular.

Capítulo 3

Los primeros pasos: Variables, input, let y print

Un programa es una lista de instrucciones que su computador puede llevar a cabo. La pregunta que surge ahora es: ¿qué clase de instrucciones se pueden usar en un programa de ordenador? Es obvio que órdenes como "ve y hazme una taza de té", son demasiado ambiguas para que algo que no es humano pueda entenderlas. Las instrucciones que demos al computador deben ser precisas. Deben especificar exactamente qué hay que hacer y con qué hay que hacerlo, esto último es quizás menos evidente. En otras palabras; una instrucción de computador dice a éste qué hacer y con qué debe hacerlo. En este capítulo veremos los **elementos** más simples de BASIC y algunas cosas muy sencillas que puede usted hacer con ellos.

Variables

El concepto de **variable** es el más importante de los que componen la programación. Una variable es una zona de la memoria del computador que se emplea para almacenar información. Si usted va a almacenar información en una zona de la memoria, necesitará tener alguna forma de referirse a ella. ¡Va a tener que darle un nombre! Realmente, no es una idea tan fuera de lo corriente, si piensa en otras formas, más tradicionales, de guardar información. Por ejemplo, cada expediente de un fichero tiene generalmente un nombre que le distingue de lo demás. ¡Piense en la confusión que supondría buscar un expediente en un fichero, si varios de aquellos tuviesen el mismo nombre! Lo mismo ocurre en BASIC. Un área de memoria que se emplea para almacenar información, una variable, debe tener un nombre único que pueda usarse para referirse a ella. Sólo hay una dificultad adicional en BASIC, y es que deberá usted definir

también qué clase de objetos va a almacenar en el área de memoria. Una de las razones para esto (veremos otras más adelante) es que la cantidad de memoria gastada para almacenar información depende del tipo de ésta. Por el momento, el único tipo de información que guardaremos en la memoria serán números de cualquier clase. Las variables que se usan para almacenar números se llaman **variables numéricas** o **variables simples.** No podrá dar a una variable cualquier nombre que se le ocurra, pues eso conduciría a una confusión por parte del ordenador. Por ejemplo, suponga que dio el nombre "1" a una variable. ¿Cómo podría saber el Spectrum cuándo se refiere a la variable en sí y cuándo al número "1"?

En el caso del Spectrum, puede usted dar a una variable simple un nombre de cualquier longitud siempre que comience con una letra y que contenga únicamente letras y dígitos. Podrá utilizar mayúsculas y/o minúsculas, pero el Spectrum no distinguirá entre ambos tipos (es decir, la variable "a" sería la misma que la variable "A", por lo que al computador se refiere). También puede intercalar espacios en cualquier lugar del nombre para hacerlo más legible, pero el Spectrum lo ignorará. Veamos algunos ejemplos de nombres correctos de variables:

suma, SUMA, Suma
Total, TOTAL, TOTAL
Este es el nombre más largo que se ha utilizado alguna vez
Total2
total3
dia2mes3ano80

Observe que todas las versiones de "suma" son tratadas como el mismo nombre, y lo mismo cuenta para "Total".

A menudo, resulta difícil inventar nombres para las variables que sugieran la clase de información que se va a guardar en ellas, pero es conveniente hacerlo. Si vuelve a leer un programa que escribió hace mucho tiempo, tener nombres claros y evidentes para las variables puede hacer mucho más sencilla la tarea de recordar cómo funcionaba su propio programa. Aunque lleve a la práctica el consejo anterior, debe evitar poner nombres demasiado largos a las variables (puede resultar muy pesado tener que escribirlos una y otra vez en un programa).

Algunos ejemplos de nombres no permitidos en el Spectrum son:

Nombre	Razón de que sea incorrecto
1dia	empieza por un número
*fecha	empieza por *, que no es una letra
fecha*	contiene * que no es letra ni dígito
respuesta?	? no es dígito ni letra
dia-fecha	- no es ni dígito ni letra

Almacenando cosas en las variables: LET

Ahora que sabemos lo que son las variables y darles un nombre, es hora de descubrir cómo almacenar información en ellas. Esto puede hacerse en BASIC usando el comando LET. Por ejemplo:

```
1Ø LET total=36
```

Esta sentencia almacenará el número 36 en una zona de memoria llamada "total". Si usted recuerda, las líneas de programa se introducen con **números de línea** que indican su orden dentro de la lista de instrucciones que componen el programa.

Este ejemplo es, de hecho, nuestro primer programa. Si lo introduce exactamente como está escrito, no ocurrirá nada hasta que pulse RUN, momento en que el Spectrum empezará a obedecer la lista de comandos. En este caso hay sólo un comando que, además, es muy sencillo de ejecutar: el número 36 se almacena en la zona de memoria llamada "total". Si reflexiona sobre ello, se dará cuenta de que algo más ha ocurrido. Antes de que el programa se ejecutase, no existía ninguna zona de memoria llamada "total" que pudiese almacenar el número 36. Cuando el Spectrum encuentra el nombre de una variable en la que quiere guardar algo, primero comprueba si ya existe, y si no es así, reserva un área de su memoria del tamaño adecuado, y recuerda su nuevo nombre.

Por ello, esta única inocente línea de programa tiene doble efecto: crear una variable llamada "total" y después guardar en ella el número 36.

Averiguando lo que hay en una variable: PRINT

El ejemplo con una sola línea, que vimos en la sección anterior, es un poco decepcionante, pues tenemos que creernos,

sin verlo, que el Spectrum ha almacenado realmente el número 36 en una variable llamada "total".

Lo que necesitamos es un comando que ordene al Spectrum buscar la variable, e imprimir lo que contiene en la pantalla del televisor. El comando BASIC que tiene este efecto se denomina PRINT (que es: "imprime", en inglés). Si añade una nueva línea, con el número 2Ø, al ejemplo anterior, tendrá el siguiente programa.

```
1Ø LET total=36
2Ø PRINT total
```

Ahora escriba RUN y ejecute el programa para comprobar, con alegría, cómo se imprime el número 36 en la parte superior de la pantalla de su televisor, en el lado izquierdo.

Es importante que en este momento comprenda exactamente qué es lo que ocurre cuando ejecuta este programa de dos líneas. Más adelante, cuando el BASIC sea como una segunda lengua para usted, entenderá qué está ocurriendo en los programas, sin ni siquiera pararse a pensar sobre ellos; pero, por ahora, es demasiado fácil leer este programa de dos líneas y creer que lo entiende, porque **parece** bastante correcto y razonable.

Así es que resumamos lo que ya hemos aprendido. La primera línea crea una variable llamada "total" y almacena 36 en ella; la segunda línea busca la zona de memoria de nombre "total", e imprime en la pantalla lo que hay guardado en ella (vea que para un principiante es fácil pensar que PRINT total lo que haría sería imprimir la palabra "total" en la pantalla. De modo que si ha entendido ya por qué es incorrecta esa interpretación, ¡ya no es usted un principiante!).

Para que la sentencia PRINT funcione, el Spectrum debe poder encontrar la variable a que aquella se refiere. Si por alguna razón intenta imprimir una variable que no ha sido creada previamente, el Spectrum le dará, con toda la razón del mundo, un mensaje de error. Para ver esto último, borre la línea 1Ø del programa anterior (únicamente debe poner 1Ø y pulsar ENTER para lograrlo) y ejecute ahora el programa (o sea, la línea 2Ø).

Debería ser capaz de saber por qué obtiene un mensaje de error:

"variable not found"

(variable no encontrada), en la parte inferior de la pantalla. ¡Este ha sido su primer error en un programa!

Aritmética

Nuestros programas se van haciendo, paulatinamente, más interesantes; pero aún falta mucho para que sean útiles. Ahora ya podemos almacenar números en las variables e imprimir el contenido de cualquiera de éstas; pero ¿y qué? Para que eso sea de alguna utilidad, debemos tener medios para cambiar el contenido de una variable y luego imprimir algo que podamos ver como una respuesta. La clave para esto está en el concepto de **expresión aritmética.**

Una expresión aritmética no es más que un cálculo aritmético cuyo valor final aún no se ha calculado. Por ejemplo, 3+6 es una expresión aritmética cuyo valor final es 9. Se pueden escribir expresiones aritméticas a la derecha del signo igual de las sentencias LET, lo que provocará que el Spectrum evalúe la expresión y almacene su resultado final en la variable correspondiente.

Pruebe, por ejemplo, con:

```
1Ø LET total=3+6
2Ø PRINT total
```

Verá como se imprime 9 en el extremo izquierdo de la parte superior de la pantalla. Como prometimos, el Spectrum ha calculado el valor final de la expresión y lo ha almacenado en "total".

Como en la mayoría de las cosas que hacen los ordenadores, hay unas reglas estrictas que definen una expresión como correcta. Se pueden usar las cuatro operaciones, que ya conocerá, de la aritmética simple: la suma y la resta se indican con los símbolos de costumbre "+" y "−", y la multiplicación y división, en cambio, utilizan "*" y "/", respectivamente.

La razón de usar * para indicar la multiplicación en vez del "aspa" normal, es que éste sería muy fácil de confundir con la letra "X".

Algunos ejemplos de expresiones aritméticas correctas son:

Expresión	*Resultado*
3+2	5
3*2	6
6/2	3
3+2−4	1
2.1+3.3	5.4

Además de las cuatro operaciones aritméticas básicas hay otras dos que pueden usarse en el Spectrum: el **signo menos unario** y la **elevación a una potencia.** Aunque "menos unario" suena algo raro, no es más que el signo de restar normal, pero utilizado delante de un único número. Por ejemplo, el "−" en 3−2 es el símbolo de restar corriente; pero el "−" en −3 es el menos unario. Aunque en ambos casos se emplea el mismo símbolo, como veremos después, se tratarán de una forma ligeramente diferente.

El símbolo de elevación a una potencia es ↑. Por ejemplo: 2↑2 se lee como dos elevado al cuadrado, es decir, cuatro. El símbolo de elevación a potencias no se usa muy a menudo, y lo mencionamos aquí más por dar una visión completa que por su importancia.

Entendiendo las expresiones: Orden de los cálculos

Parece que la idea de una expresión aritmética resulta muy clara. Pero hay una complicación oculta. Por ejemplo, si escribe la expresión 3+2*4 "aparentemente inocente", puede significar: tres más dos (o sea, 5) por cuatro (respuesta veinte) o, tal vez signifique: tres más el resultado de dos por cuatro (o sea, tres más ocho cuya respuesta es once). Puede resultar extraño que haya dos formas posibles de calcular el resultado de una misma expresión, y que usted piense que una de las dos maneras es la correcta y la otra, evidentemente, no lo es. Pero resulta que, incluso en aritmética, no hay respuestas absolutas. La interpretación correcta es una convención y no un mandato divino. La cuestión de si debemos hacer primero la suma o el producto en una expresión como 3+2*4, se ha establecido por un acuerdo general. Según esta convención, el producto es más importante que la suma, por tanto debe efectuarse primero, con lo que el valor correcto de nuestro ejemplo es once.

Este acuerdo de que el producto es más importante que la suma, puede formalizarse en términos de asignar **prioridades** a cada operación, de forma que las operaciones más prioritarias se efectúan primero.

La asignación de prioridades puede extenderse a toda operación que pueda formar parte de las expresiones (incluso algunas que aún no hemos visto). Las prioridades que usa el Spectrum,

para conocer el orden en que debe realizar los cálculos aritméticos, son:

Operación	*Prioridad*
↑	10 - la más alta
−unario	9
*,/	8
+,−	6 - la más baja

El motivo de que la prioridad más alta sea 10 y la más baja 6, es permitir asignar prioridades a otros operadores que aún no hemos tratado.

Para evaluar una expresión, siempre deberá ejecutar primero las operaciones de prioridad más alta. Si en una expresión dos operadores tienen la misma prioridad, entonces debe evaluar primero el que está más a la izquierda, es decir, en ausencia de otro criterio, los cálculos se ejecutan siempre de izquierda a derecha (siempre que tengan igual prioridad, claro).

Puede que todo esto parezca un poco complejo a la hora de calcular una expresión aritmética simple, pero es necesario para poder escribir expresiones sin ambigüedad. No obstante, hay otro modo de especificar el orden de evaluación que anula las prioridades establecidas: el uso de paréntesis (). Un criterio aceptado desde hace muchísimo tiempo, establece que cualquier parte de una expresión encerrada entre paréntesis debe evaluarse primero. Por ejemplo, aunque 3+2*4 era igual a 11/1, (3+2)*4, da como resultado 20. Siempre que tenga dudas acerca del orden en que el Spectrum efectuará cualquier expresión artimética, ponga paréntesis en las partes de la expresión que usted quiera que se calcule primero. En ocasiones los paréntesis requieren un gasto de tiempo adicional, pero nunca causarán problemas.

Variables y constantes: Expresiones completas

Hasta ahora hemos visto expresiones aritméticas que solamente usaban números, pero no hay ningún inconveniente para que incluyamos variables en dichas expresiones.

Si escribimos una expresión del tipo "total+3", el Spectrum buscará la variable llamada "total", recuperará el valor que esté almacenado en ella, y luego sumará 3 a ese número. Por ejemplo, si "total" tiene almacenado 32, la expresión "total+3" daría

como resultado 35. Dese cuenta de que no hemos dicho nada acerca de que lo que estaba almacenado en "total" se altere de manera alguna. Unicamente se utiliza el valor de la variable para calcular la expresión.

Un número como 32 se denomina una **constante** (porque su valor nunca cambia), y ahora sabemos ya que las expresiones pueden constar de variables, constantes y operadores aritméticos como ↑, +, *, /, −, uniendo aquellas. El resultado de una expresión siempre es una constante, y esta constante se puede almacenar en una variable por medio de la sentencia LET.

Un programa corto

Ahora podemos escribir un programa corto que sume dos números, usando todo lo que hemos aprendido hasta el momento sobre las variables y las constantes:

```
1Ø LET numero1=23.34
2Ø LET numero2=44.34
3Ø LET respuesta=numero 1+numero 2
4Ø PRINT respuesta
```

Si introduce el programa y lo ejecuta, verá que la suma de los dos números de las líneas 1Ø y 2Ø aparece en la pantalla como resultado de la línea 4Ø. Aunque éste es un ejemplo muy sencillo, presenta una gran cantidad de ideas de programación.

En las líneas 1Ø y 2Ø se usa la, ahora familiar, sentencia LET para almacenar dos constantes en dos variables; en la línea 3Ø se evalúa la expresión "numero1+numero 2", y su resultado se guarda en una tercera variable: "respuesta". La línea 4Ø imprime el contenido de "respuesta" en la pantalla. Si piensa que es sencillo, ¡vamos bien!

Pruebe a cambiar las líneas 1Ø y 2Ø para sumar números diferentes, y la línea 3Ø para obtener el valor de otras expresiones aritméticas.

Otra forma de cambiar las variables: INPUT

En el ejemplo anterior, las dos variables "numero1 y numero 2" contenían números que habíamos almacenado en ellas por

medio de sentencias LET. Esto está bien pensado, salvo si queremos usar el programa muchas veces con valores diferentes. Como ya le sugerimos, la única forma posible de cambiar los valores que guardan las variables es editar cada línea antes de ejecutar el programa. Está claro que necesitamos una sentencia que nos permita introducir cualquier valor en las variables mientras **que el programa se está ejecutando.** Para esto se utiliza la sentencia BASIC: INPUT.

Por ejemplo, pruebe el programa siguiente:

```
10 LET numero1=5
20 INPUT numero2
30 LET respuesta=numero1+numero2
40 PRINT respuesta
```

Al ejecutar esto, podría sorprenderse al ver que no ocurre nada. ¡No se asuste! Lo que pasa es que se ejecutó la línea 1Ø, y "5" se guardó en la variable "numero1". Tras eso, el Spectrum pasó a la línea 2Ø, donde obedeció el comando INPUT y éste le indicó que esperase a que usted le introdujese un número, motivo por el cual no ocurre nada en este momento. El Spectrum está esperando a que teclee un número y pulse después ENTER para indicarle que ha terminado de introducir/corregir el número. En ese instante, él almacenará el número que haya tecleado en la variable "numero2", y seguirá ejecutando la instrucción siguiente. Por tanto, si todavía no lo ha hecho, ejecute el programa y escriba un número que le guste. Le agradará ver cómo aparece su número, más cinco unidades, en el sitio de costumbre de la pantalla.

Ya tenemos ahora dos formas de almacenar números en las variables: LET e INPUT. Es fundamental entender la diferencia entre la forma de trabajar de ambas. Como en el caso de LET, si una variable no existe antes de utilizarla en una sentencia INPUT, el Spectrum la creará. Si usted quiere almacenar siempre el mismo valor o el resultado de una expresión en una variable, lo mejor es que use LET; si, en cambio, quiere almacenar valores diferentes en una variable cada vez que ejecute el programa, la sentencia INPUT es la más apropiada.

Constantes y variables en expresiones

Una de las características más poderosas de BASIC, es que puede usar una expresión casi en cualquier parte donde pueda usar una constante o una variable. Por ejemplo, en una sentencia PRINT puede escribir:

```
30 PRINT numero1+numero2
```

y el Spectrum calculará la expresión e imprimirá directamente el resultado.

También se cumple que la forma más simple de una expresión es que se componga de una sola variable o constante. Se puede decir, como ejemplo, que el número 3 es una constante, o bien una expresión llevada a su forma más sencilla. Por todo lo anterior, no sólo puede usar una expresión en cualquier sitio donde podría usarse una constante o una variable, sino que además puede usar una constante o una variable allá donde podría usar una expresión, es decir:

```
LET numero1=numero2
```

y

```
PRINT 3
```

son ambas sentencias BASIC correctas.

Descripción del BASIC

Es difícil describir cualquier lenguaje, y BASIC no es una excepción. La dificultad estriba en que, mientras que es sencillo poner ejemplos de lo que es correcto, es difícil exponer todas las variaciones posibles.

Al comienzo de este capítulo hemos introducido la sentencia LET con:

```
LET total=36
```

pero esto no indicaba, en modo alguno, que se pudiese escribir algo como

```
LET total=numero
```

o

```
LET total=numero1+numero2
```

Para intentar superar esta dificultad es corriente dar una definición general que indique los tipos de cosas que permite hacer una sentencia. Por ejemplo, la forma general de la sentencia LET puede poner así:

LET "variable simple"="expresión aritmética"

donde las entidades encerradas entre comillas no se toman al pie de la letra, sino que serán reemplazadas por un ejemplo del tipo establecido. Entonces, en una sentencia LET real, "variable simple" será sustituida por un nombre de variable como total, suma, número, etc.

A través del libro, las sentencias BASIC serán presentadas con un ejemplo, y luego las definiremos de la misma forma que LET en este caso. A medida que aprendemos más sobre una sentencia, puede revelarse como necesario redefinirla incluyendo una gama mayor de características. Hasta el momento, las otras dos sentencias comentadas han sido PRINT e INPUT. Podemos definir ambas como:

PRINT "expresión aritmética"

o

INPUT "variable simple"

pero, como ya veremos, no son sus definiciones finales.

INPUT con comentarios

A pesar de que podemos usar INPUT para almacenar información en las variables, la forma en que el Spectrum se detiene y espera a que alguien introduzca un número puede resultar poco satisfactoria. Lo que se requiere es la capacidad de imprimir en la pantalla un mensaje que diga algo así: "introduzca ahora un número", o bien: "cual es su número". Esos mensajes a menudo se denominan **mensajes de entrada («input prompt»,** en inglés), y BASIC proporciona dos métodos parecidos para imprimir dichos mensajes.

Pruebe el programa siguiente:

```
10 PRINT "esto es un mensaje"
20 "cual es su numero"; numero1
```


La línea 1Ø presentará: "esto es un mensaje", en la parte superior izquierda de la pantalla, y la línea 2Ø escribirá: "cual es su numero" en la parte inferior, y después esperará a que se introduzca un número. En los dos casos, los caracteres impresos en pantalla son los que aparecen encerrados entre comillas. Un conjunto de caracteres entre comillas se conoce con el nombre de **literal** (literal string), o simplemente string. Puede utilizar tanto INPUT como PRINT para producir mensajes de entrada en pantalla, y decidir entre una u otra dependerá de lo que sea más conveniente en cada caso. Consideremos, para aclarar esto, la versión siguiente del programa de suma de dos números que ya vimos:

```
10 PRINT "cual es su primer numero"
20 INPUT numero1
30 INPUT "cual es su segundo numero"; numero2
40 PRINT numero1+numero2
```

PRINT combinado

La capacidad de imprimir mensajes en pantalla, es, evidentemente, una característica muy útil para otras cosas, aparte de los mensajes de entrada. Por ejemplo, en el programa anterior habría sido mejor imprimir un mensaje diciendo que el número que aparecía en la pantalla era la suma de los números introducidos. De hecho, usted puede usar una sentencia PRINT única para imprimir varias cosas al mismo tiempo. Si cambia la línea 4Ø del programa previo por:

```
40 PRINT numero1; "+"; numero2; "="; numero1+numero2
```

comprobará que se escribe el contenido de "numero1", después, un espacio y el signo más seguido por otro espacio, después, el contenido de "numero2" seguido por el signo igual, con un espacio a cada lado y, por último, la respuesta.

Se puede considerar esta sentencia PRINT como una lista de informaciones que se van a escribir en pantalla, separada cada información por un punto y coma, y que aparecerá en la siguiente posición de impresión libre.

Podemos actualizar la definición general de PRINT, que será ahora:

PRINT "lista de impresion"

donde "lista de impresión" es una lista de componentes separados por punto y coma. Los componentes pueden ser expresiones o literales (strings). Cada sentencia PRINT comienza la impresión de una nueva línea. En el capítulo siete volveremos a ver la definición de "lista de impresión", y la ampliaremos para que incluya medios de dar un formato a la información de la pantalla. Pero, hasta entonces, esta versión de "lista de impresión" es suficiente para satisfacer nuestros propósitos.

Algunos programas de muestra

Incluso con la pequeña parte de BASIC aprendida hasta el momento, es posible escribir algunos programas útiles, aunque sencillos. Un ejemplo: si usted quisiera saber a cuántos dólares equivale una cantidad de pesetas dada, podría serle de utilidad un programa de conversión de moneda. La forma que debe tener este programa, en líneas generales, es fácil de exponer como sigue:

Pregunta al usuario el valor de conversión
Pregunta el número de pesetas que se quieren convertir a dólares
Imprime el número de pesetas multiplicado por el valor de conversión

Ya debería saber que este programa es muy sencillo para su Spectrum; por ello, antes de mirar la versión que viene a continuación, intente escribir su propia versión. Recuerde incluir mensajes de entrada y comentarios aclaratorios. No existe una forma perfecta de escribir un programa; así pues, no se preocupe si su versión es diferente de la que aquí aparece: ¡puede que sea mejor!

```
10 PRINT "Conversion de pesetas a dolares"
20 INPUT "¿Cual es el valor de conversion?"; valor
30 INPUT "¿Cuantas pesetas quiere cambiar en dolares?";
cantidad
```



```
40 PRINT
50 PRINT "Con"; cantidad; "pesetas"
60 PRINT "puede adquirir"; valor*cantidad; "dolares"
```

La línea 10 escribe únicamente un título para el programa; las 20 y 30 preguntan y aceptan los valores de entrada necesarios. Fíjese en cómo se usa la línea 40 para escribir una línea en blanco, y separar, así, la salida del programa en una sección de entrada y otra con el resultado. Las líneas 50 y 60 escriben la respuesta con una explicación de su significado.

Considere, como un nuevo ejemplo, el problema de calcular la distancia de detención de un coche que circula a cualquier velocidad en millas/hora. El principal problema para escribir el programa es saber cómo se calcula la distancia de frenado. Es importante señalar en esta etapa que ningún computador puede calcular nada, a menos que usted sepa explicarle **cómo** hacer dicho cálculo. En el código de circulación, leemos que la distancia de detención consta de dos componentes: distancia de reacción y distancia de frenado. La distancia de reacción del conductor es, aproximadamente, en pies, la velocidad en millas/k; la de frenado es un poco más difícil, y se calcula como el cuadrado de la velocidad dividido por 20. Sin embargo, ambas cantidades son muy fáciles de calcular para el Spectrum, por lo que, para que el programa resulte más gráfico, es razonable imprimir las distancias de reacción, frenado, distancia total de detención y a cuántas veces la longitud del automóvil equivale esta última. Este último cálculo se basa en la estimación de que un coche corriente (en Inglaterra) mide 14 pies de largo. El problema es ahora muy fácil de escribir:

```
10 PRINT "distancia de detencion"
20 PRINT
30 INPUT "Velocidad en millas/hora="; velocidad
40 PRINT velocidad; "millas/hora"; "A"
50 PRINT
60 PRINT "Distancia de reaccion="; velocidad; "pies"
70 LET distfren=velocidad*velocidad/20
80 PRINT "Distancia de frenado="; distfren; "pies"
90 PRINT "Distancia total="; velocidad+distfren; "pies"
100 PRINT
110 PRINT "Lo que equivale a"; (velocidad+distfren)/14;
"longitudes de coche"
```

La línea 1Ø imprime el título del programa; la 3Ø escribe un mensaje pidiendo la única información de entrada necesaria aquí: la velocidad en millas/hora. Esta se almacena en la variable "velocidad". Las líneas 4Ø y 5Ø comienzan a dar la respuesta imprimiendo la velocidad para la que se efectúa el cálculo, y dejan una línea en blanco. La línea 6Ø imprime la distancia de reacción, que no requiere ningún cálculo, pues numéricamente es la misma que la velocidad. La distancia de frenado la calcula la línea 7Ø y se imprime en la 8Ø, mientras que la línea 9Ø calcula la distancia total y la escribe (vea que es un ejemplo del uso de expresiones en sentencias PRINT). Por último, 11Ø calcula la distancia de detención en longitudes de autos. Ponga atención y fíjese en cómo se usan los paréntesis en 11Ø, para asegurarse de que la suma se efectúa primero.

El ejemplo final de este capítulo es de tamaño considerablemente mayor, y realmente útil si le interesa la construcción. Un problema muy común es el de estimar cuánta arena, agregado y cemento tendría que comprar para fabricar una plancha de hormigón. Le ayudará a resolver esta difícil tarea el programa siguiente:

"Calculos de hormigon"

De nuevo, su primer problema es saber cómo hacer los cálculos antes de comenzar con el programa mismo. En los libros de construcción se dice que un metro cúbico de hormigón compuesto de una parte de cemento, x parte de arena é y partes de agregado (es decir, una mezcla 1:x:y) necesita:

$$\frac{1}{0.025*(1+x+y)}$$

sacos de cemento,

$$\frac{x*1.5}{(1+x+y)}$$

metros cúbicos de arena y

$$\frac{y*1.5}{(1+x+y)}$$

metros cúbicos de agregado (no nos importa demasiado comprender a qué se deben estas ecuaciones. A menudo ocurre que un programa se escribe con información que el programador no comprende del todo: ¿por qué debería ser de otra manera?). Convertir esta información en un programa es, de nuevo, relativamente fácil. La primera parte del programa deberá pedir las dimensiones de la plancha de hormigón y después calcular su volumen. El programa tendrá que pedir además la relación de la mezcla, y, usando las ecuaciones anteriores, averiguar el número de sacos de cemento y el volumen de arena y agregado que se necesitan. Finalmente, los resultados deberán imprimirse de forma que el usuario los encuentre razonables. Los detalles del programa son:

```
1Ø PRINT "Calculador de hormigon"
2Ø PRINT
3Ø INPUT "¿Que grosor, en mm?" gros
4Ø INPUT "¿Que longitud, en m?"; lon
5Ø INPUT "¿Que anchura, en m?"; ancho
6Ø PRINT
7Ø LET vol=gros*.ØØ1*lon*ancho
8Ø PRINT "Volumen total="; vol; "metros cubicos"
9Ø PRINT
1ØØ INPUT "¿Cuantas partes de arena por una de cemento?"; partarn
11Ø INPUT "¿Cuantas partes de agregado por una de cemento?"; partagr
12Ø LET total=1+partarn+partagr
13Ø LET cemento=vol/total/Ø.Ø25
14Ø LET arena=vol*partarn*1.5/total
15Ø LET agreg=vol*partagr*1.5/total
16Ø PRINT "Usando una mezcla 1:"; partarn; ":"; partagr
17Ø PRINT "necesitara:"
18Ø PRINT cemento; "sacos de cemento"
19Ø PRINT arena; "metros cubicos de arena"; "y"
2ØØ PRINT agreg; "metros cubicos de agregado"
```

Las líneas 3Ø-8Ø piden las dimensiones de la plancha y calculan e imprimen el volumen total. Las líneas 1ØØ-11Ø preguntan la relación de la mezcla y la 12Ø calcula 1+x+y, que se usará en cálculos sucesivos. Las líneas más interesantes de todo el programa son 13Ø-2ØØ. Las que abarcan el intervalo 13Ø-15Ø,

llevan a cabo los cálculos primordiales; las que ocupan 16Ø-2ØØ, escriben los resultados. Si observa con cuidado los cálculos, verá que una expresión aritmética BASIC no siempre se parece a la ecuación de la que procede. Por ejemplo, podría usted caer en el error de escribir:

$$\frac{1}{0.025*(1+x+y)} \quad \text{como} \quad 1/0.025*(1+x+y)$$

pero el resultado de esta expresión aritmética BASIC sería el de dividir 1 por 0.025, y después multiplicar el resultado de eso por (1+x+y), o sea, ejecutaría:

$$\frac{1}{0.025}*(1+x+y)$$

en vez de la ecuación que queríamos. La expresión correcta es:

1/0.025/(1+x+y)

o si quiere usar paréntesis para verlo más claro:

1/(0.025*(1+x+y))

En la sección que imprime los resultados, puede ver cómo la respuesta calculada está **intercalada** con los mensajes escritos. No olvide que usted no tiene que escribir programas que den resultados de una manera formalizada: experimente y trate de obtener los resultados de manera que sean claros y naturales.

Capítulo 4
Selección y repetición: El flujo de control

Hemos escrito, hasta ahora, programas que eran una lista de instrucciones que se ejecutaban secuencialmente desde la primera a la última, y de una en una. Aunque es posible escribir programas útiles sin usar nada más, los programas sólo se hacen realmente interesantes cuando se puede cambiar el orden de ejecución de las instrucciones.

El flujo de control

Si observa cualquiera de los programas de ejemplo del final del capítulo tres, podrá seguir con su propio dedo el orden en que se ejecutan las instrucciones. Puede ver esto como una representación del **flujo de control** del programa. Cada instrucción gobierna o **controla** por turno lo que está realizando el Spectrum, y después **cede el control** a la instrucción siguiente. En ausencia de cualquier otra información, la **instrucción siguiente** pasa a ser la línea que está inmediatamente debajo. Entonces, el **flujo de control por defecto** es una línea que empieza en la primera instrucción del programa, y termina en el final de éste. El siguiente programa, muy simple, aclara esta condición de defecto.

```
1Ø INPUT a
2Ø PRINT a
3Ø INPUT b
4Ø PRINT b
```

Fig. 4.1.

Bucles: Sentencia GOTO

El BASIC proporciona una única sentencia para cambiar el flujo por defecto (flujo normal) de control: la sentencia GOTO. Pruebe el programa siguiente:

```
1Ø LET test=Ø
2Ø PRINT test
3Ø GOTO 2Ø
```

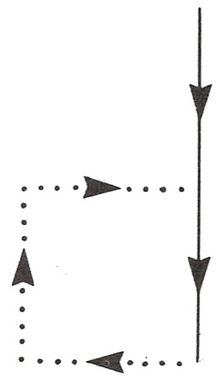


Fig. 4.2.

Si ejecuta este programa, verá cómo la pantalla se llena de ceros (uno en cada línea) y después, el Spectrum se detiene y pregunta: "Scroll? y/n".

Si le responde con y (o sea, sí) a esta pregunta, la pantalla se volverá a llenar con ceros otra vez, y así sucesivamente. Este programa es nuestro primer ejemplo de un **bucle** o **lazo.** El dibujo del flujo de control del programa explica rápidamente por qué es tan apropiada la palabra **bucle.** Al comienzo, la línea 1Ø almacena un cero en la variable "test"; después la 2Ø imprime el contenido de "test". La línea 3Ø es un novedad, puesto que usa GOTO ("ve a", en inglés), pero su significado se deduce claramente de la traducción de esa frase inglesa.

La sentencia GOTO 2Ø hace que el Spectrum tome como instrucción siguiente a la número 3Ø, la número 2Ø. Por lo tanto, el control pasa, después de ejecutar la línea 3Ø, a la línea 2Ø, y el contenido de la variable "test" se imprime en la pantalla una vez más. Después de eso, otra vez se ejecuta la línea 3Ø. Eso devuelve el control a la línea 2Ø, etc., hasta que la pantalla se llena con ceros y el Spectrum pide permiso para **desplazarla hacia arriba** y dejar sitio para más información, que en este caso serían más y más ceros. La repetición de las líneas 2Ø y 3Ø continuará siempre, y el diagrama de flujo de control muestra que éste consiste en un bucle. A pesar de todo, éste no es un bucle muy útil, y, puesto que el único modo de detenerlo es pulsando la tecla

BREAK o contestando "n" a la pregunta de "Scroll?", normalmente se denomina **bucle infinito.**

La sentencia GOTO se puede emplear para obligar al Spectrum a que ejecute cualquier instrucción. Su forma general es:

GOTO "expresion aritmetica"

con lo que puede escribir algo como:

GOTO 2*1Ø

(que significa lo mismo que GOTO 2Ø). Pero cosas como esas no suelen usarse muy a menudo, y es mejor pensar en GOTO como una sentencia de la forma:

GOTO "numero de linea"

A lo mejor se está preguntando qué pasaría si escribiese algo como GOTO 5Ø, y resultase que la línea 5Ø no existe. La mayoría de las versiones de BASIC detendrían el programa y le darían un mensaje de error, indicando que ha intentado pasar el control a una línea que no existe; pero el Spectrum tiene una versión de BASIC un poco distinta. Si no existe el número de línea al que intenta pasar el control, entonces dicho control pasará a la línea que tenga el número superior más cercano al de la línea inexistente. Por ejemplo, si utiliza GOTO 5Ø y la línea 5Ø no existe, pero sí que existe la línea 55, el control pasará precisamente a ella. Si no existe una línea con un "número mayor" más próximo al que aparece en GOTO, lo que ocurre es que GOTO intentará enviar el control fuera del programa, y éste se detendrá, **pero sin dar mensajes de error.** Esto significa que es prácticamente imposible obtener un error con GOTO. Tal vez eso le parezca una ventaja; pero ¡tenga mucho cuidado!, porque si ha cometido algún error al poner el número de sentencia en GOTO, el Spectrum **no le dirá** que usted ha pasado el control a una línea que no existe. Puede que, al principio, esto no diese problemas, porque la línea de número mayor más próximo fuese precisamente aquella a la que quería transferir el control después de todo; pero si más tarde insertase cualquier línea(as) de programa, surgirían problemas aparentemente inexplicables.

El programa de ejemplo del bucle infinito de antes, nos ha servido para presentar el concepto de transferencia de control a

puntos diferentes de un programa; pero, en realidad, no indica la clase de cosas para las que se emplean los bucles.

Una idea importante de programación es la repetición de un conjunto de operaciones. Esto recibe el nombre de **iteración.** Ejemplo: la instrucción LET cantidad=cantidad+1 únicamente suma una unidad al contenido de "cantidad", y el resultado lo vuelve a almacenar en ésta. En otras palabras, **incrementa** en una unidad el contenido de "cantidad". Si usted repite esta operación usando GOTO, tendrá algo más que añadir 1 a una variable: ¡tiene un programa que cuenta! Pruebe lo siguiente:

```
1Ø LET cuenta=Ø
2Ø LET cuenta=cuenta+1
3Ø PRINT cuenta
4Ø GOTO 2Ø
```

Verá cómo se llena la pantalla con los números de 1 a 21 y aparece la pregunta de costumbre, "Scroll?". Respondiendo "s" a esto, puede seguir viendo números durante mucho tiempo.

Dese cuenta de que, añadiendo un GOTO a una instrucción que suma 1 a una variable, lo que obtiene es un programa que hace un poco más que contar. De hecho, hemos generado una **secuencia de números.** Esto responde a la filosofía de la programación real mucho más que cualquiera de los ejemplos del capítulo anterior. Pruebe el programa siguiente para ver cómo una estructura de bucle hace algo un poco más útil:

```
1Ø LET cuenta=Ø
2Ø PRINT "x="; cuenta; "x al cuadrado="; cuenta*cuenta
3Ø LET cuenta=cuenta+1
4Ø GOTO 2Ø
```

Este programa imprimirá una lista de números y del cuadrado de dichos números.

Aun estando muy familiarizado con la idea de bucle, puede que se desoriente cuando trata de saber el valor de una variable en cualquier punto de un bucle. Por ejemplo, en el caso del programa (con un lazo) para contar, que ya vimos, el valor de "cuenta" que se escribía era uno; pero en el programa de los cuadrados era, en cambio, cero. Esta diferencia se debe únicamente al **lugar** donde el programa tiene la línea que añade una unidad a "cuenta", y, además, al **valor inicial** que se da a

"cuenta" antes de que comience el bucle. Si cambia la línea 1Ø del programa de los cuadrados por: 1Ø LET cuenta=1, entonces será 1 el primer valor que se imprima.

Entender lo que ocurre en el interior de los lazos de programa, se vuelve más fácil con la práctica; aunque es, al fin y al cabo, una cuestión de seguir el curso del programa claramente y sin precipitarse.

Selección y condiciones: Sentencia IF

Por más que GOTO sea una sentencia muy útil, la única cosa que puede hacer con ella es crear bucles infinitos. Eso es muy conveniente para programas simples como imprimir tablas de valores, pero es algo insuficiente para muchas aplicaciones.

Nos está faltando una sentencia que detenga el lazo cuando se cumpla una cierta **condición.** Suponga que queremos escribir un programa que sume diez números y después nos dé la respuesta. Hasta el momento, lo mejor que podemos hacer es teclear los números de uno en uno, y sumarlos a un total **acumulado** que imprimiremos cada vez, como hace el siguiente ejemplo:

```
1Ø LET total=Ø
2Ø LET cuenta=Ø
3Ø INPUT "numero="; numero
4Ø LET total=total+numero
5Ø LET cuenta=cuenta+1
6Ø PRINT cuenta; "total="; total
7Ø GOTO 3Ø
```

Para detener en cualquier momento el programa, introduzca STOP cuando éste le pida otro número. Si ejecuta el programa, verá que imprime mucha más información de la que usted quería obtener. Lo que se pretendía era leer 1Ø números, irlo acumulando en una variable, o imprimir el resultado final Esto puede lograrse usando la sentencia IF:

```
1Ø LET total=Ø
2Ø LET cuenta=Ø
3Ø INPUT "numero="; numero
4Ø LET total=total+número
```

```
5Ø LET cuenta=cuenta+1
6Ø IF cuenta=1Ø THEN GOTO 8Ø
7Ø GOTO 3Ø
8Ø PRINT cuenta; "total="; total
```

La única diferencia entre este programa y el primero que dimos para sumar 1Ø números, reside en que este último utiliza la sentencia IF de la línea 6Ø. Cada vez que se ejecuta el bucle formado por las sentencias de 3Ø a 7Ø, se obedece la sentencia IF. Esto hace que se compare el contenido de la variable "cuenta" con 1Ø. Si no coinciden (o sea, cuenta no es 1Ø), el control pasará a la siguiente sentencia detrás del IF, que aquí es la 7Ø, y el GOTO que va detrás de THEN no tiene ningún efecto. Sin embargo, cuando "cuenta" sea igual a 1Ø, se ejecutará el GOTO 8Ø que va tras de THEN, y el control se transmitirá a la sentencia 8Ø que imprimirá la respuesta, con lo que el bucle se termina (y en este caso, también el programa).

Hay muchas formas de usar IF... THEN GOTO para alterar el flujo de control en función de que se cumpla o no una **condición** dada. Antes de poder continuar investigando la clase de cosas que pueden hacerse con IF, tenemos que saber qué tipos de condiciones está permitido emplear.

Todas las condiciones permitidas en una sentencia IF tienen una forma muy simple:

"expresion aritmetica 1" "relacion" "expresion aritmetica 2"

Ya sabemos qué es una expresión aritmética. Por tanto, el único elemento nuevo es: **relación.**

Hay seis tipo de relación en BASIC:

relación	*significado*
=	igual a
>	mayor que
<	menor que
<=	menor o igual que
>=	mayor o igual que
<>	distinto de

Fíjese en que en el Spectrum cada uno de estos símbolos se obtiene pulsando una única tecla. Obtendrá mensaje de error si introduce un "<" y un ">" para formar un único "<>". El significado de cada una de estas relaciones en BASIC es el normal en matemáticas.

A menudo se denomina a las condiciones **expresiones condicionales,** y una condición es como una expresión que hay que evaluar, pero en este caso sólo hay dos resultados posibles: verdadero o falso. Es importante leer correctamente las condiciones para evitar confusiones. Así, la condición "cuenta=3" **no es** una instrucción que dé el valor 3 a la variable "cuenta", sino una **pregunta** sobre el contenido de "cuenta". Si lo que hay almacenado en "cuenta" es un 3, entonces "cuenta=3" es verdadero (true, en inglés), y si no es igual a 3, "cuenta=3" tiene valor falso (false, en inglés). Como vemos, el Spectrum utiliza el signo "=" de dos formas distintas: como una instrucción para almacenar valores en variables, y como una relación en una condición. La única forma de saber el significado correcto en cualquier caso es mirar el resto de la instrucción. Ejemplos de condiciones son:

cuenta<>4	falso, si cuenta contiene 4, y cierto en otro caso
cuenta*2>10	cierto, si cuenta*2 es mayor que 10, y falso en otro caso
3>1	SIEMPRE verdadero
1>6	SIEMPRE falso

Para hacer patente la idea de que una condición es una expresión que tiene un resultado de entre dos posibles (verdadero o falso), será mejor decir que el Spectrum representa ambos valores como números: verdadero se representa como 1, y falso como Ø. En otras palabras, si una condición es verdadera, su valor es 1, y si es falsa, su valor es Ø. Comprobemos que eso funciona realmente así, mediante este programa:

```
1Ø INPUT "primer numero"; a
2Ø INPUT "segundo numero"; b
3Ø PRINT "el resultado de"; a; ">"; b; "es"; a>b
4Ø GOTO 1Ø
```

Verá que se imprime 1 ó Ø, dependiendo de que "a>b" sea verdadero o falso para los números que usted introduce. Podría sorprenderle que se pueda escribir una condición en una sentencia PRINT, donde normalmente se escribe una expresión arit-

mética. Eso es únicamente un efecto de que las condiciones son iguales que las expresiones aritméticas y pueden usarse en cualquier parte donde se usan aquellas. Es más, se pueden mezclar condiciones y expresiones aritméticas sin ningún problema, siempre que ponga atención al orden en que se van a evaluar unas y otras: las relaciones tienen prioridad 4, lo que significa que cualquier cálculo aritmético se efectúa antes de evaluar dichas relaciones. Por ejemplo:

PRINT (1>2)+(3>1)+(3=2+1)

darán en pantalla un 2, porque (1>2) es falso y vale 0, (3>1) es cierto y vale 1, y (3=2+1) es cierto y vale 1, lo que nos daría 0+1+1=2 (vea cómo las condiciones llevan paréntesis para evitar que el Spectrum evalúe primero los operadores aritméticos). Esta clase de cosas es una característica avanzada de BASIC. Por ello, no se preocupe si no lo entiende del todo: lo más importante es que comprenda que algo como 2=3, es una expresión y tiene un valor verdadero o falso, y no es ninguna clase de instrucción para realizar cálculos.

Nos hemos desviado un poco del tema de la sentencia IF para examinar el concepto de expresión condicional, pero, con esta nueva información, el resto de este capítulo debería parecer más sencillo. La forma general de la sentencia IF ... THEN GOTO es:

IF "expresión condicional" THEN GOTO "numero de linea"

Si la expresión condicional da 1 como resultado, es decir: es verdadera, se ejecutará la sentencia GOTO que va después de THEN. Si la expresión condicional toma valor Ø, es falsa, se ejecutará la sentencia siguiente al IF.

Existen tantas formas de usar "IF ... THEN GOTO", aparte de salir de los bucles, que es difícil dar ejemplos de todas ellas. Pero si ha comprendido las ideas de flujo de control y de cómo pueden usarse IF y GOTO para cambiarlo, no tendrá dificultad en comprender los ejemplos que hay en el resto del libro.

Empleo de IF

La única manera de saber lo útil que resulta la sentencia IF, es escribir sus propios programas utilizándola. De ese modo irá

aprendiendo lentamente todas las formas corrientes de variar el flujo de control, dependiendo del resultado de una expresión condicional. No obstante, para acelerar este proceso y evitar un uso inadecuado de la sentencia IF, podría serle de ayuda ver ejemplos de los casos más comunes en que se usa esta sentencia.

Una sentencia IF se puede emplear para **saltarse** una sección de programa:

```
1Ø INPUT a
2Ø IF a>Ø THEN GOTO 4Ø
3Ø LET a=-a
4Ø PRINT a
```

Esta clase de programa rechaza los números negativos que usted introduzca. La sentencia IF de la línea 2Ø mira si el número "a" es mayor que cero, y, si es así, el control pasa a la línea 4Ø (saltándose, pues, la línea 3Ø). Si resulta negativo o nulo, se ejecutará la sentencia 3Ø, que cambia el signo del contenido de "a". En otro programa sería pasada por alto otra serie de instrucciones, dependiendo de alguna condición diferente. El **aspecto** de estos "saltos" de trozos de programas, está reflejado en la figura 4.3, que muestra cómo el resultado de una condición decide si un conjunto de instrucciones se va a ejecutar o se va a pasar por alto.

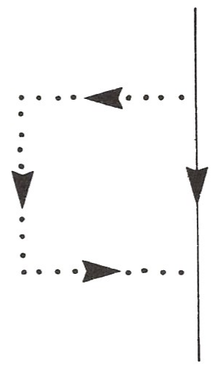


Fig. 4.3.

Una extensión de la idea de saltar líneas BASIC, es elegir entre dos listas de comandos. En este caso, es más fácil entender el concepto después de mirar la forma que tiene el flujo de control (ver figura 4.4).

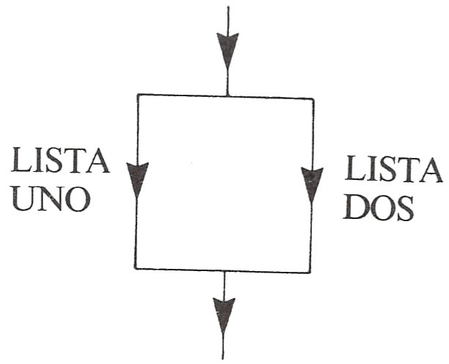


Fig. 4.4.

Lo que se va a ejecutar depende de la condición empleada en la sentencia IF. Es útil ver la figura 4.4 antes de examinar un ejemplo de sentencia IF que **seleccione** entre dos conjuntos de instrucciones, porque en BASIC resulta un poco difícil distinguir este tipo de división en dos alternativas. Considere el programa siguiente:

```
1Ø INPUT a
2Ø IF a<Ø THEN GOTO 6Ø
3Ø PRINT "a es positivo"
4Ø LET b=a
5Ø GOTO 8Ø
6Ø PRINT "a es negativo"
7Ø LET b=-a
8Ø PRINT a,b
```

De acuerdo con el valor de "a" se ejecutarán las líneas 3Ø, 4Ø, y 5Ø, o bien, 6Ø y 7Ø. **El punto de división** del diagrama anterior corresponde a la sentencia IF de la línea 2Ø. El **punto de reunión** es la línea 8Ø, ya que ésta es la primera sentencia que se ejecutará, sea cual sea la lista de instrucciones seleccionada. Si intenta hacer corresponder el diagrama de la figura 4.4 con el programa en sí, verá que, aunque aquel representa lo que ocurre de verdad, es difícil casar el uno con el otro. Es una consecuencia de que en BASIC resulta imposible escribir dos listas de instrucciones alternativas de forma que queden una cerca de la otra. Por este motivo, es mejor representar el control con el diagrama:

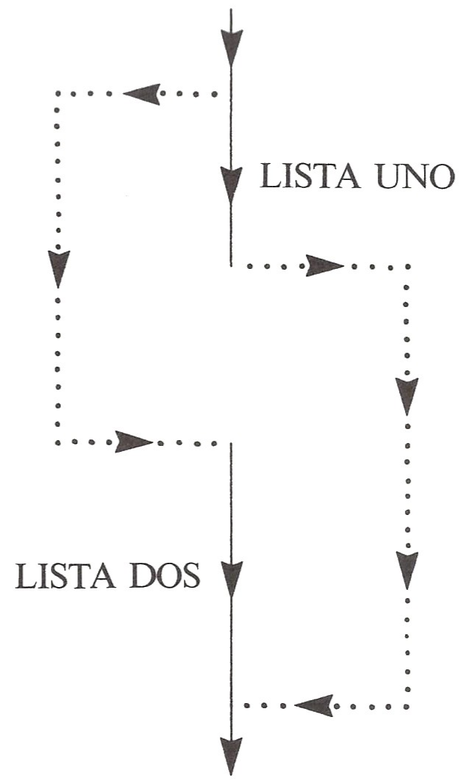


Fig. 4.5.

Si mira la figura 4.5, se dará cuenta de que es una forma "estirada" de la figura 4.4. El uso de IF para seleccionar entre dos alternativas, es más sencillo de entender con la figura 4.4; pero la figura 4.5 se aproxima más a la realidad.

Ya conocíamos el otro uso importante de la sentencia IF: salir fuera de los bucles. El flujo de control para este caso puede verse en el diagrama de la figura 4.6.

En él puede observar la forma familiar de un bucle infinito. El camino que conduce fuera de él, de regreso al flujo normal de control, corresponde a un GOTO ejecutado cuando la condición en la sentencia IF es verdadera. Tenga en cuenta que el punto en que la sentencia IF provoca la salida del lazo puede colocarse en cualquier parte. En otras palabras: se puede salir de un bucle en cualquier parte de éste, desde su primera sentencia a la última.

Por ejemplo:

```
1Ø LET a=Ø
2Ø LET a=a+1
3Ø IF a=2Ø THEN GOTO 6Ø
```

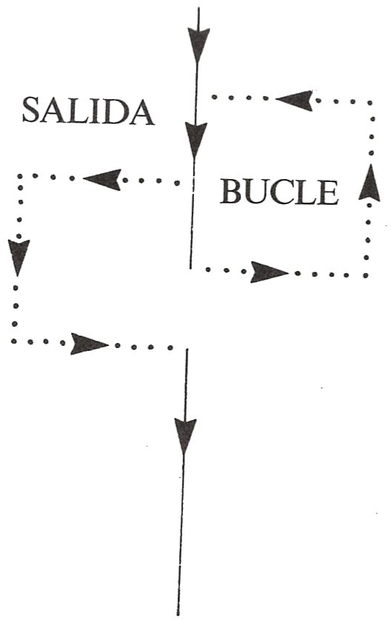


Fig. 4.6.

```
4Ø PRINT a
5Ø GOTO 2Ø
6Ø PRINT "terminado"
```

tiene el punto de salida en medio y

```
1Ø LET a=Ø
2Ø LET a=a+1
3Ø PRINT a
4Ø IF a=2Ø THEN GOTO 6Ø
5Ø GOTO 2Ø
6Ø PRINT "terminado"
```

tiene su punto de salida al final del lazo.

En general, aunque es posible colocar en cualquier parte el punto de salida de un bucle, es mejor situarlo justo al principio, o bien al final. Decimos esto, porque es mejor evitar siempre salir a mitad de un bucle (técnicamente, un lazo con el punto de salida colocado al principio se denomina "do-while", y con el punto de salida al final: "do-until". Sin embargo, ambos nombres vienen de otros lenguajes de programación, y no nos interesan para aprender BASIC).

Los bucles que tienen su punto de salida al final, pueden

simplificarse cambiando la condición por la contraria. Veámoslo:

```
1Ø LET a=Ø
2Ø LET a=a+1
3Ø PRINT a
4Ø IF a=2Ø THEN GOTO 6Ø
5Ø GOTO 2Ø
6Ø PRINT "terminado"
```

puede ser convertido en el programa equivalente,

```
1Ø LET a=Ø
2Ø LET a=a+1
3Ø PRINT a
4Ø IF a<>2Ø THEN GOTO 2Ø
5Ø PRINT "terminado"
```

En el primer ejemplo, tenemos que el lazo termina cuando "a=2Ø", y la línea 5Ø contiene el GOTO que cierra el bucle cuando a<2Ø. En el segundo caso, hemos cambiado la condición del IF de antes por su contraria, lo que nos permite eliminar la sentencia 5Ø del ejemplo primero. Ahora, el bucle continúa cuando a<>2Ø, por medio de GOTO 2Ø del IF.

Una vez que haya visto esta clase de cosas un par de veces, será capaz, en cada caso, de adoptar la forma más corta y correcta sin pensarlo apenas. De cualquier forma, es mejor recordar, de vez en cuando, por qué se hacen así las cosas.

Como comentamos al principio de este apartado, hay muchas maneras de emplear una sentencia IF para alterar el flujo de control de los programas, y no debe sentirse limitado a aquellas que se han presentado aquí. Pero es un buen consejo decirle que no experimente, porque cuantas más formas distintas tenga que emplear IF, mayor será la dificultad que tendrá para entender sus propios programas.

En esencia, la regla es hacer que el diagrama de flujo de control de programa sea lo más sencillo posible, de modo que sus programas sean fáciles de leer, entender y depurar.

La sentencia FOR

Dejando a un lado la situación del punto de salida, los bucles difieren en otros aspectos: exactamente en dos. Todos los lazos

continúan ejecutándose hasta que se satisface una condición; pero en muchos casos esto equivale a ejecutar dicho lazo un número fijo de veces. Así, si quiere escribir "Hola" en la pantalla cinco veces, podría usar esto:

```
1Ø LET a=1
2Ø PRINT "Hola"
3Ø LET a=a+1
4Ø IF a<=5 THEN GOTO 2Ø
```

Pero ocurre que esta situación es tan común, que BASIC proporciona dos sentencias más (FOR y NEXT) que permiten repetir las listas de instrucciones con mayor facilidad. Si usamos FOR y NEXT para poner cinco veces "Hola", el nuevo programa quedaría con este aspecto:

```
1Ø FOR a=1 TO 5
2Ø PRINT "Hola"
3Ø NEXT a
```

Es muy sencillo deducir del programa lo que significan FOR y NEXT. La variable "a" se utiliza para contar el número de veces que debe ejecutarse el lazo (de la misma manera que en el primer ejemplo). La diferencia es que ahora todo se realiza automáticamente. La sentencia FOR pone primero "a" a 1. Cada vez que se encuentra NEXT en el programa, se añade 1 a la variable "a", y mientras ésta no sea **mayor** que 5, el control vuelve a la sentencia 2Ø (o sea, hay un GOTO 2Ø implícito en esta nueva estructura). El resultado es que la sentencia PRINT de la línea 2Ø se ejecuta cinco veces, antes de que el control pase a la sentencia que vaya a continuación de NEXT.

La forma general del buble FOR ... NEXT es:

```
FOR "variable indice"="valor inicial" TO "valor final"
    .  .  .  .  .
    .  .  .  .  .
NEXT "variable indice"
```

La primera vez que se encuentra esta estructura en un programa, se asigna a "variable índice" el "valor inicial". Cada vez que se alcanza NEXT, se incrementa en 1 la "variable indice" y, mientras el valor de ésta no exceda el "valor final", el control se

transfiere a la sentencia que viene detrás del FOR. La única restricción es que el nombre de "variable índice" debe ser de una sola letra (por tanto, "variable índice" podrá ser cualquiera de las letras de la "a" a la "z"). Para asegurarse de que comprende exactamente cómo funciona el lazo FOR, pruebe estos ejemplos:

```
1Ø FOR i=1 TO 1Ø
2Ø PRINT i
3Ø NEXT i

1Ø FOR z=1ØØ TO 11Ø
2Ø PRINT z
3Ø NEXT z
```

En general, tanto "valor inicial" como "valor final" pueden ser expresiones aritméticas completas, pero es importante resaltar que **sólo** se evalúan una vez: al **principio del bucle.** Esto será más claro si piensa en esta sentencia como si FOR se ejecutase una sola vez, al principio del bucle. El valor de "variable indice" puede usarse en cualquier expresión aritmética dentro del bucle, pero **no debe cambiarse nunca su valor.** En otras palabras: puede usar la "variable indice" en la parte derecha de cualquier sentencia LET, es decir, a la derecha del signo "=", pero no puede ponerla a la izquierda. Por ejemplo, este programa imprimirá una tabla de multiplicar:

```
1Ø INPUT "¿Que tabla desea ?"; t
2Ø INPUT "¿En que numero empieza ?"; s
3Ø INPUT "¿En que numero acabo ?"; e
4Ø FOR i=s TO e
5Ø PRINT i; "x"; t; "="; i*t
6Ø NEXT i
```

Vea cómo los valores inicial y final del FOR son expresiones aritméticas (o mejor, variables simples) en este caso. Observe también el uso de la "variable indice" en la línea 5Ø.

La sentencia FOR simple es muy útil, pero existe una forma de ésta ligeramente más sofisticada que, ocasionalmente, es más útil aún, y que es mejor que conozcamos. En la sentencia FOR (la que hemos visto ahora), cada vez que se ejecuta el bucle entero, siempre se añade 1 a la "variable índice". Esto es válido cuanto está usando dicha variable para contar el número de

veces que se ejecuta el lazo. Algunas veces ocurre que los cálculos que se efectúan dentro del lazo FOR, necesitan una variable, cuyo valor cambia más de una unidad cada vez que se pasa por el bucle. Estos casos se manejan en BASIC utilizando la sentencia STEP junto con el FOR. La forma genérica es:

FOR "variable indice"="valor inicial" TO "valor final" STEP "incremento"

El "incremento" indicado después de STEP es la cantidad que se añade a la variable índice, cada vez que se pasa por el bucle. Luego, el FOR simple que vimos sería equivalente a poner aquí STEP 1. La única cosa que debe cuidar cuando use FOR-STEP es asegurarse bien de que sabe cuándo va a terminar el bucle. La regla es que éste terminará cuando el valor de "variable índice" **sea mayor** que el valor final. O sea, en un lazo FOR, el índice nunca puede ser más grande que el "valor final".

Veamos esto en acción con los programas:

```
10 FOR a=.4 TO 10 STEP .01
20 PRINT a
30 NEXT a
```

o

```
10 FOR a=1 TO 100 STEP 25
20 PRINT a
30 NEXT a
```

El valor del incremento que sigue a STEP puede ser negativo, en cuyo caso sería mejor hablar de **decremento.** Por ejemplo:

```
10 FOR a=100 TO 0 STEP -15
20 PRINT a
30 NEXT a
```

Puede usted encontrar ligeras dificultades en averiguar cuándo terminan éste y otros bucles similares. Sin embargo, nuestra regla es casi la misma que para incrementos positivos: Cada vez que se pasa por el bucle, el valor de la variable índice se decrementa en lo que indica la expresión tras el STEP, y el bucle

finalizará cuando el valor del índice sea más pequeño que el "valor final".

Al igual que para los incrementos, es posible tener "valor final" o "valor inicial" negativos, y esto puede empezar a complicar las cosas. Considere este corto programa:

```
10 FOR a=-100 TO 50 STEP 10
20 PRINT a
30 NEXT a
```

A primera vista parece que no tiene sentido; pero, si lo piensa detenidamente, verá que realmente se aplican las mismas reglas. El valor del incremento se suma a la variable índice en cada ejecución del bucle, hasta que el valor de ésta sea mayor que el "valor final", si el incremento es positivo, o hasta que sea menor, si el incremento es negativo.

Uso del bucle FOR

Hay una o dos reglas que rigen el uso de los lazos FOR. Como usted puede usar cualquier sentencia válida de BASIC dentro del FOR-NEXT, también es posible usar un GOTO o un IF para salir del lazo FOR antes de que termine (o sea, antes de que la "variable índice" tome el "valor final"). Esto es permitido por el BASIC del Spectrum, pero no por muchas otras versiones del lenguaje, las cuales darían error al dejar incompleta una sentencia FOR. Por esa razón es mejor no acostumbrarse a programar de esa forma "poco ortodoxa".

Muy a menudo se usan sentencias FOR combinadas, o, mejor, "anidadas". Si comprendió bien el uso de la sentencia, no habrá nuevos problemas para entender cosas como ésta:

```
10 FOR a=1 TO 10
20 FOR b=1 TO 10
30 PRINT a,b
40 NEXT b
50 NEXT a
```

que es absolutamente correcta, ya que el FOR constituido por las líneas 20, 30 y 40, está **totalmente** incluido dentro del lazo FOR que se extiende entre las líneas 10 y 50. Esto significa que

el bucle **más interno** se ejecuta completo cada vez que se pasa por el bucle **externo.** Es muy fácil cometer una "ligera" equivocación y terminar así:

```
1Ø FOR a=1 TO 1Ø
2Ø FOR b=1 TO 1Ø
3Ø PRINT a,b
4Ø NEXT a
5Ø NEXT b
```

lo cual dará un mensaje de error.

Sentencia IF con «:»

Hay una versión de IF más general aún que la que conocemos hasta ahora. Al igual que podíamos incluir GOTO detrás de la parte THEN de la sentencia, en realidad podemos usar cualquier comando BASIC válido:

```
1Ø INPUT a
2Ø IF a<Ø THEN PRINT "es negativa"
3Ø IF a=Ø THEN PRINT "es cero"
4Ø IF a>Ø THEN PRINT "es positiva"
5Ø GOTO 1Ø
```

Lo más sencillo es ver esto como una versión de IF más simple que la de saltar a otras sentencias. En este caso, la instrucción que sigue a THEN se ejecuta sólo si la condición de IF es verdadera. No se puede añadir mucho más a esta descripción, salvo hacer notar que a THEN puede seguirle cualquier sentencia válida en BASIC, ¡incluso otro IF!

Esta extensión de IF parece muy útil al principio; pero pronto llegará usted a la conclusión de que pocas veces querrá ejecutar una sola instrucción, si una condición se verifica. Sería mucho mejor poder incluir toda una lista de instrucciones detrás de THEN. El Spectrum permite eso, aunque es importante decir que no todos los BASIC lo hacen. Usted puede agrupar una serie de sentencias en una línea, separando una de otra con los dos puntos ":", y se ejecutarán una a una en un orden de izquierda a derecha. Por tanto:

```
1Ø PRINT "Primera": LET a=1: PRINT "segunda"; a
```


se ejecuta como si fuese una única línea de BASIC en orden de izquierda a derecha; es decir, sería equivalente a:

```
1Ø PRINT "Primera"
2Ø LET a=1
3Ø PRINT "Segunda "; a
```

El uso de ":" para agrupar sentencias en la misma línea, amplía la regla del flujo de control normal. Ahora, en lugar de obedecerse las instrucciones por orden creciente de números de línea (o sea, de arriba a abajo, según el listado de pantalla), las instrucciones que se agrupan juntas en una línea se ejecutan de izquierda a derecha. Este comportamiento es exactamente el mismo que tiene una persona que lea instrucciones escritas en una hoja de papel.

Aunque esta extensión simplifica mucho las cosas, debería recordar que, como otras versiones BASIC no la tienen, usarla "más de la cuenta" hará que usted dependa bastante del BASIC del Spectrum cuando programe. Como ya hemos indicado, el lugar donde resulta realmente útil esta facultad de poder agrupar comandos en una sola línea, es precisamente a continuación de THEN en la sentencia IF. Por ejemplo, al final de la mayoría de los programas de juegos es corriente preguntar si el jugador desea seguir o dejar de jugar. En la rutina siguiente, que podría servir como parte final de algún programa, se espera una respuesta del operador en forma "1" (para decir sí) o "Ø" (para decir "no"). Cualquier otro valor introducido se toma como erróneo, y se imprime un mensaje que indica la equivocación y pide una vez más la respuesta:

```
11Ø INPUT "Quiere volver a jugar (si=1, NO=Ø) "; a
12Ø IF a=1 THEN GOTO xxx
13Ø IF a<>Ø THEN PRINT "Debe contestar con 1 o Ø":
GOTO 11Ø
14Ø PRINT "ADIOS"
```

Nótese el empleo ":" para agrupar dos comandos juntos en la línea 13Ø. Las xxx de 12Ø deberían ser reemplazadas, por supuesto, por el número de línea donde comenzaría el juego.

Hay otra sentencia BASIC útil que no hemos comentado, que puede usarse junto con IF. Supóngase que como resultado de una condición desea detener el programa, cosa que hasta

ahora sólo sabemos hacer poniendo GOTO y un número de línea que esté al final de todo el programa. La sentencia BASIC:STOP se puede poner en cualquier parte del programa, para terminar éste y devolver el control al usuario:

```
10 PRINT "¿Quiere seguir?: si=1, no=0"
20 INPUT a
30 IF a=0 THEN PRINT "Adios": STOP
40 IF a<>1 THEN PRINT "No entiendo eso. Debe contes-
tarme     con 1 o 0": GOTO 10
50 PRINT "Entonces sigamos, maestro"
60 GOTO 10
```

Vea cómo los dos puntos agrupan juntos los comandos en las líneas 30 y 40, y la utilización del STOP en la línea 30.

Ejemplo final

Como último ejemplo de las sentencias IF, FOR, consideramos de nuevo el programa de la distancia de frenado que dimos en el capítulo 3, pero modificándolo para que sea una especie de "concurso". Se preguntará al jugador por las distancias de reacción, frenado y detención total, en pies y para una gama de velocidades dada.

```
10 PRINT "Distancia de detencion"
20 LET marca=0
30 FOR v=10 TO 80 STEP 10
40 PRINT "Cual es la distancia de reaccion a "; v; " mph?"
50 INPUT r
60 IF r=v THEN LET marca=marca+1
70 PRINT "Cual es la distancia de frenado "; v; " mph?"
80 INPUT f
90 IF f=v*v/20 THEN LET marca=marca+1
100 PRINT "¿Cual es la distancia total de detencion?"
110 INPUT d
120 IF d=v*v/20=v THEN LET marca=marca+1
130 NEXT v
140 PRINT
150 PRINT "Ha logrado: "; marca; " puntos"
160 PRINT "de un maximo posible de 24"
```

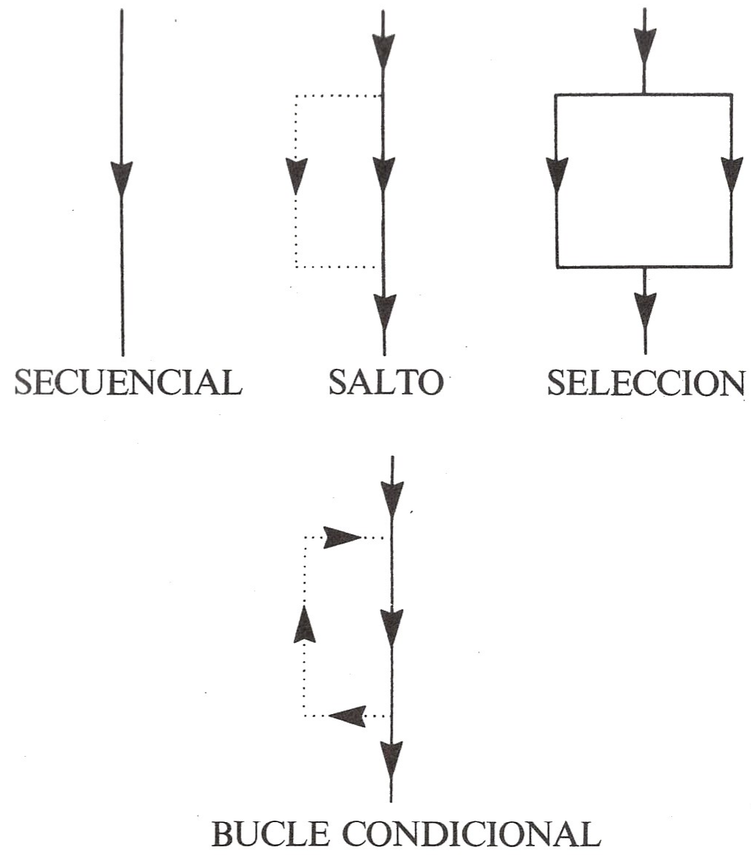



Fig. 4.7. Diagramas de flujo de control.

Es difícil saber exactamente qué hay en la mente de un programador experimentado. Sea lo que sea, hay un "algo" que marca la diferencia entre un principiante y un experto. Una cosa que parece cierta es que, además del catálogo de programas que hemos visto antes y una colección de manejables "astucias", las formas de los diagramas de flujo de control que hemos estado manejando en este capítulo, estarán siempre presentes. Será mejor mencionar en este momento que ya se ha demostrado (matemáticamente) que cualquier programa se puede construir usando únicamente el flujo secuencial (por defecto), la selección y el bucle condicional. Esta cuestión es tan importante, que volvemos a presentar juntos los diagramas de flujo de control, para que le sea más fácil recordarlos en adelante.

Capítulo 5
Números y textos

Hasta ahora, los únicos programas que hemos escrito, siempre han usado números. Si eso fuese todo lo que los computadores pudieran hacer, se diferenciarían muy poco de las calculadoras de bolsillo. Veremos en este capítulo cómo el Spectrum puede manejar textos y caracteres con la misma sencillez que los números y dígitos. En la última parte del capítulo, presentaremos algunas otras formas de ampliar las operaciones que podemos hacer con los datos: los "arrays" y su almacenamiento en cinta.

Cadenas de caracteres

En el capítulo 3 introdujimos la idea de **cadena de caracteres** (string) como un conjunto de letras encerradas entre comillas, que nos permitían escribir mensajes y comentarios en la pantalla. Verdaderamente, lo que hemos venido llamando "cadena" era en realidad una **constante de cadena.** El emplear la palabra constante ya le indicará que existen cosas como **variables de cadena.** Una variable de cadena es similar a una variable simple, ya que ambas son áreas de memoria que pueden usarse para almacenar información. En este caso, la información es un conjunto de caracteres en lugar de un número. Las reglas para dar nombres a estas nuevas variables son distintas de las que vimos al hablar de variables simples. El nombre de una variable de cadena consiste en la única letra seguida del signo del dólar ($). Dicho signo se emplea para distinguir las variables simples de las variables de cadena; o sea, "b" es una variable simple, pero "b$" es una variable de cadena y, por tanto, diferente a la primera.

Se puede usar la sentencia LET para almacenar una constante de cadena en una variable de cadena, y estas últimas pueden imprimirse con la sentencia PRINT:

```
10 LET a$="esto es una cadena"
20 PRINT a$
```

En realidad, una variable de cadena puede utilizarse en cualquier sitio donde pueda hacerlo una variable simple, siempre que tenga sentido. Así, puede usar INPUT a$ para introducir una cadena desde el teclado cuando un programa se está ejecutando. Pero LET total=3+a$ no tendrá, evidentemente, ningún sentido (no se pueden sumar una cadena de caracteres y un número). Fíjese en la diferencia que hay, en particular, entre:

```
10 LET a$="1"
```

y

```
20 LET a$=1
```

La línea 10 es correcta, puesto que el 1 aparece entre comillas y, por lo tanto, sería una cadena. Pero la línea 20 dará un error, ya que a$ es una variable de cadena, y 1 es un número.

La aparición de variables de cadena es apasionante, pues nos abre toda una serie de posibilidades, para manejar textos e incluso diálogos con el Spectrum. Hasta ahora, sin embargo, la única clase de programas que sabemos escribir es:

```
10 INPUT "Cual es su nombre"; n$
20 PRINT "Hola! "; n$; "Soy tu computador Spectrum"
```

Lo cual, aun estando bien como comienzo, puede derivar en diálogos poco naturales como:

```
Cual es su nombre? Pedro Gomez
Hola! Pedro Gomez. Soy tu computador Spectrum
```

El problema es que podemos usar INPUT y PRINT para introducir y escribir cadenas, o usar LET para almacenarlas, pero no tenemos forma de cambiar esas cadenas almacenadas. Esto es como si tuviésemos INPUT, PRINT y LET para los números, pero no pudiésemos hacer operaciones con ellos (está

claro que sólo podríamos escribir unos cuantos programas limitados). La respuesta está en inventar una "artimética" para las cadenas, de forma que podamos tener "expresiones de cadena".

Expresiones de cadena

Antes de contar las posibilidades del Spectrum para manejar cadenas, sería mejor considerar qué clase de cosas podríamos querer hacer con ellos, y después ver si el Spectrum puede realizarlo. Si un programa tuviese el nombre de alguien en n$, y su apellido en a$, sería muy útil poder unir ambos para lograr una cadena mayor. A esa unión de cadenas se la denomina: **concatenación.** Otra cosa que sería de desear es la capacidad de **extraer** una parte de una cadena. Por ejemplo, se podría extraer su apellido de una cadena que contuviese el nombre completo; por ejemplo, extraer "Perez" de "Pedro Perez".

Una cadena que está incluida en otra cadena más grande, se denomina **subcadena.** Podríamos querer reemplazar una subcadena con otra, dentro de una cadena dada. Así, por ejemplo, si quisiese guardar en secreto el apellido de "Pedro Perez", podría cambiar "Perez" por asteriscos, para obtener "Pedro*****". Por último, otra gran ventaja sería poder detectar la aparición de una determinada subcadena dentro de una cadena mayor, como sería poder detectar "Perez" en la cadena "Pedro Perez" que, suponemos, estaba guardada en n$.

Recapitulando, las operaciones con cadenas que nos harían falta son la concatenación, extracción de subcadenas, reemplazo de subcadenas por otras y búsqueda de subcadenas en las cadenas mayores.

Nuestro primer deseo, la concatenación, lo satisface inmediatamente el Spectrum, pues cuenta con el signo + para realizarla. Si a$ contiene "abcd" y b$ contiene "efgh", después de la sentencia:

```
LET c$=a$+b$
```

la variable c$ contendrá "abcdefgh". Dese cuenta de que tenemos ya dos funciones para el símbolo +: una, como suma corriente de números, y otra, como el signo de concatenación de cadenas.

Puede usar el signo "+" varias veces en una misma expresión:

```
LET c$="Sr. "+f$+" "+1$
```

Esa sentencia concatenará las cuatro cadenas que aparecen, de forma que, si f$ contiene el nombre "Pedro", y 1$ el apellido "Perez", entonces c$ contendrá: "Sr. Pedro Perez". Hay que usar un espacio en blanco, como puede ver, para evitar que c$ pase a contener "Sr. PedroPerez".

Extraer o cambiar una subcadena puede hacerse usando una única operación nueva: **troceado** (slicing). La mayor parte de las otras versiones de BASIC usan métodos mucho más complicados que el **troceado de cadenas del Spectrum.**

Aunque el Spectrum pueda ser algo diferente cuando maneja las cadenas, el hecho es que ofrece unas enormes ventajas, en estas operaciones, sobre el resto de las máquinas en general. Un troceador de cadena especifica una subcadena, dando la posición de la primera y la última de sus letras. Por ejemplo:

```
PRINT "12345678" (3 TO 6)
```

pondrá en la pantalla la subcadena "3456" (o sea, la subcadena que comienza con el tercer carácter y termina con el sexto de la cadena grande). Recuerde que TO debe introducirse, una vez más, con una sola pulsación (TO se encuentra en la tecla "F". Si teclease "T" y después "O" por separado, el troceado no funcionaría:

La forma general de un troceador de cadena es:

"Expresion de cadena"	"Expresion aritmetica 1"	TO	"Expresion aritmetica 2"

y la subcadena especificada por esta construcción es la que comienza con el carácter que marca la "expresión aritmética 1" (las expresiones van en el orden de aparición de los caracteres que se toman) y termina con el carácter que ocupa la posición dada por la "expresión aritmética 2".

Puede parecer que esto es una cosa muy complicada, y, en realidad, va más allá de las formas usadas en otras versiones BASIC para trocear cadenas, lo que es una ventaja. La mayoría de las veces, "expresión de cadena" es una constante o una

variable de cadena, y las expresiones aritméticas son también números constantes o variables numéricas simples del tipo que aparecen en:

"abcd" (2 TO 3)

o

"abcd" (cuenta TO 3)

Sin embargo, ya debe haberse dado cuenta de que una de las características principales de BASIC es que, en cualquier lugar donde se puedan usar una constante o una variable, se pueden usar también expresiones. El troceado de las cadenas no es una excepción a esta regla, y permite escribir cosas como:

("abcd"+"efgh") (principio TO principio+3)

lo cual, primero concatena las dos cadenas "abcd" y "efgh", para formar una única "abcdefgh", y después extrae de esta última cuatro letras, empezando por la que ocupe la posición guardada en "principio".

Escribir expresiones de cadena complicadas no debe preocuparle más de lo que le preocuparía escribir expresiones aritméticas complicadas: ¡las dos son posibles!

Hay una serie de casos particulares de la notación del troceado que es mejor que conozcamos. El comienzo y el final de una cadena son empleados tan a menudo para formar subcadenas, que al especificar el troceado puede dejar en blanco la "expresión aritmética 1", indicando esto que la subcadena comienza con el mismo carácter que la cadena; si no existe la "expresión aritmética 2", se supone que la subcadena terminará con el mismo carácter que la cadena de la que se obtiene. De ese modo:

"123456" (TO 5) significa "123456" (1 TO 5) que es "12345"
"123456" (3 TO) significa "123456" (3 TO 6) que es "3456"
"123456" (TO) significa "123456" (1 TO 6) que es "123456"

También hay una abreviatura muy útil que extraerá un único carácter de cualquier posición de una cadena. En vez de tener

que escribir (n TO n) para extraer el n-simo carácter (el que ocupa la posición n) de la cadena, basta con que escriba (n):

"123456" (3) significa "123456" (3 TO 3) que es "3"

Es posible obtener cosas erróneas usando este mecanismo. Por ejemplo: si usted especifica la posición de un carácter que no existe, provocará un mensaje de error que indica "Subscript wrong" (subíndice equivocado) o "Integer out of range" (número entero fuera de rango). En "123456" (4 TO 7) o "123456" (— TO 3) ocurrirán sendos errores. Pero, si, en cambio, usa una posición de comienzo de subcadena que es **menor** que la posición final, como "123456" (2 TO 1), no resultará ninguna clase de error. En vez de eso, el resultado es una forma muy especial de cadena: **la cadena vacía.** La cadena vacía no tiene ningún carácter y en expresiones con cadenas juega un papel muy similar al del número cero en expresiones aritméticas. La cadena constante que corresponde a la cadena vacía es "" (dos dobles comillas seguidas, sin ningún espacio entre ellas). Fíjese en la diferencia entre "" y " ". La primera es la cadena vacía y no tiene ningún carácter, mientras que la segunda es una cadena que posee un solo carácter: el espacio o blanco. Desde el punto de vista del ordenador, el espacio es un carácter como lo son las letras, y ocupa una posición de impresión en la pantalla. Además, al guardarlo, ocupa la misma memoria del computador que los demás caracteres. En cambio, imprimir la cadena nula no tiene ningún efecto: las sentencias PRINT a; b y PRINT a; ""; b, producen idénticos resultados.

Como ejemplo del uso de subcadenas podemos considerar el problema de imprimir el nombre de un mes, una vez conocido su número (es decir, para 12 imprimir Dic., etc.). Pruebe este programa:

```
10 LET y$="EneFebMarAbrMayJunJulAgsSepOctNov-
Dic"
20 INPUT "numero del mes ?"; mes
30 PRINT "El mes "; mes; " es "; y$ (1+3*(mes-1) TO
3*mes)
40 GOTO 20
```

Si introduce un número del intervalo 1 al 12, el programa escribirá la abreviatura del nombre del mes en cuestión. La

forma en que funciona esto es extrayendo las tres letras del nombre deseado de la cadena y$. La mejor manera de comprender el troceado de la línea 3Ø, es haciéndolo a mano para unos pocos valores de "mes". Si mes=4, entonces (1+3*(mes−1) TO 3*mes) sería (1Ø TO 12), y si cuenta las letras en y$, obtendrá "Abr".

Se puede emplear el troceado para cambiar los caracteres de una subcadena. Puede emplear la notación normal de troceado para definir una subcadena que debe ser cambiada a la izquierda del signo "=" de las sentencias LET. Por ejemplo:

```
1Ø LET a$="123456"
2Ø LET a$(3 TO 4)/"ABCDEFG"
3Ø PRINT a$
```

escribirá en la pantalla "12AB56". Esta propiedad es muy lógica como ampliación de la sentencia LET, usada para almacenar una cadena en una variable de cadena. El operador de troceado únicamente restringe el conjunto de caracteres que son alterados por sentencia LET. Si la cadena de la derecha del signo "=" del LET es mayor que la subcadena especificada, los caracteres sobrantes se ignoran, y si es más pequeña, entonces los caracteres que faltan se **rellenan** con blancos. Ejemplo:

```
1Ø LET a$="123456"
2Ø LET a$(2 TO 5)="ABC"
3Ø PRINT a$
```

nos dará la cadena "1ABC 6" en el televisor (vea el blanco que va después de C).

La única cosa que queda de nuestra lista inicial de peticiones para manejo de cadenas es el poder comprobar si una subcadena dada está presente dentro de una cadena mayor. El BASIC del Spectrum no proporciona un método directo para realizar esto, pero amplía el uso de las expresiones condicionales (ver capítulo 4) con las cadenas, y esto se puede utilizar para lograr los fines antes comentados de buscar subcadenas dentro de otras cadenas. Puede utilizar todas las relaciones que presentamos en su momento, con las cadenas. El significado "=" y "<>" es muy claro: dos cadenas son iguales si son de la misma longitud y contienen los mismos caracteres en el mismo orden; en otro caso serán distintas.

Pero, ¿qué significado tienen las relaciones <, >, <= y >=, cuando se aplican a las cadenas? La respuesta varía de una a otra versión de BASIC. En el caso del Spectrum están definidas de forma que: a$<b$ es cierto, si la cadena a$ va antes que b$ en orden alfabético. Lo malo es que estamos tan familiarizados con dicho orden, que nos olvidamos, a veces, de cómo funciona. Si las dos cadenas que se van a comparar son de un solo carácter, a$<b$ si la letra de a$ es anterior a la de b$ en orden alfabético. Por ejemplo: "a"<"b" es cierto, pero "d"<"b" es falso. ¿Qué ocurre cuando comparamos cadenas de un solo carácter, pero éste no es literal, como, por ejemplo, "*"<"$"? En el caso de las letras, el alfabeto nos proporciona ya un orden predefinido, y lo que podremos hacer es ampliar este ordenamiento para incluir todos los demás símbolos que puede usar el Spectrum. Dicho de otra forma: ¡necesitamos un superalfabeto! El Spectrum ya dispone de él. Si mira el Apéndice A del manual del Spectrum, verá una lista del conjunto completo de caracteres de la máquina en el orden predefinido para ellos, y ese orden es el que se emplea para decidir si a$<b$ es verdadero o no. Si usted **no tiene** el **manual del Spectrum** a mano, o si está interesado, puede imprimir todo el juego de caracteres en orden correcto, usando el programa:

```
1Ø FOR i=34 TO 255
2Ø PRINT "caracter"; i; "="; CHR$ (i)
3Ø NEXT i
```

La línea 1Ø da como comienzo el valor de 34, porque los caracteres de Ø a 33 son de "control" y no pueden imprimirse. No se preocupe por el CHR$(i) de la línea 2Ø, ya que se explicará más adelante. Si ejecuta el programa, se sorprenderá, tal vez, al ver que las palabras como PRINT y FOR se escriben en pantalla como si fuesen un carácter. Eso es consecuencia de que el Spectrum trata como un único carácter todo lo que pueda ser introducido con una sola pulsación de tecla, aunque aparezca en la pantalla como una palabra completa.

Volviendo a nuestra duda de si "*"<"$" es cierto o falso, vemos que "*" es el carácter 42, y "$" el carácter 36. Por tanto, "$" va antes que "*" en el orden del juego de caracteres, y nuestra relación "*"<"$" es falsa. Empleando igual razonamiento, puede comprobar el valor de verdad de las demás relaciones.

Si dos cadenas contienen más de un carácter, se comparan carácter a carácter hasta encontrar el primer par de caracteres diferentes en ambas. Entonces la relación que hay entre ellas (>, <, etc.) se decide en base a esos dos caracteres. Por ejemplo: "ABCD"<"AZCD" es cierto, pues el primer par de caracteres diferentes son B y Z. Como "B"<"Z" es cierto, la relación entera también.

Si una de las cadenas es igual que la otra, salvo que posee algunos caracteres más, entonces la comparación se establece de acuerdo a la longitud: "ABCD"<"ABCDE", es cierta, pues ambas cadenas no tienen caracteres diferentes al compararlas de carácter en carácter, y "ABCD" es la más corta.

La idea de longitud de las cadenas es muy importante, y por ello el Spectrum posee una forma muy fácil de averiguar la longitud de cualquier cadena. Pruebe el programa siguiente:

```
1Ø INPUT a$
2Ø PRINT LEN(a$)
3Ø GOTO 1Ø
```

verá que como resultado imprime el número de caracteres que usted teclea en respuesta a INPUT. En general:

LEN ("expresión de cadena")

dará el número de caracteres de "expresión de cadena" (el empleo de LEN se discutirá con más detalle en el capítulo 6).

Tablas

Las cadenas y números son los únicos tipos de datos que el Spectrum puede manejar, y esto es suficiente para muchos propósitos. Sin embargo, el Spectrum proporciona una forma de usar los tipos de datos básicos de una manera más sofisticada: las tablas (array, en inglés).

Considere el problema de leer cinco números y luego imprimirlos en orden inverso en la pantalla. Hasta ahora, el único medio de hacerlo es:

```
1Ø INPUT a1,a2,a3,a4,a5
2Ø PRINT a5, a4, a3, a2, a1
```


lo que no es demasiado malo en el caso de cinco números; pero piense en cómo sería el programa si el problema fuese para cien números.

Sería conveniente poder referirnos a una variable de la forma a(i), donde i puede tomar valores de 1 a 5 en el interior de un bucle FOR. En ese caso, podríamos escribir:

```
2Ø FOR i=1 TO 5
3Ø INPUT a(i)
4Ø NEXT i
5Ø FOR i=5 TO 1 STEP -1
6Ø PRINT a(i),
7Ø NEXT i
```

Esta es exactamente la forma que BASIC le permite usar. El conjunto de variables de a(1) hasta a(5) se denomina **tabla** y una variable particular a(i), recibe el nombre de **elemento** de la tabla (figura 5.1).

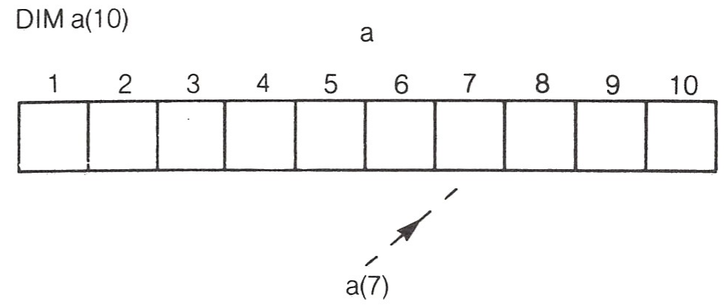


Fig. 5.1. Una tabla unidimensional.

La única complicación es que, antes de que pueda usar una tabla, deberá decirle al computador cuántos elementos va a tener esa tabla. Para ello se emplea la sentencia DIMension:

```
1Ø DIM a(5)
```

que debería añadirse al programa de antes para que funcione. Si define una tabla con cinco elementos, e intenta usar a(6), obtendrá un mensaje de error por haber tratado de usar algo que no existe. Parece, pues, tentador, definir tablas más grandes de lo necesario para evitar esos errores, pero, ¡cuidado!, las tablas llenarán rápidamente la memoria que la máquina tiene para trabajar.

Para dar nombres a las tablas existe una restricción similar a la que había para variables índices: sólo pueden tener una letra de largo. Eso significa que como máximo puede definir veintiséis tablas diferentes con nombres de "a" a "z"; pero esto es, generalmente, más que suficiente.

Cuando se usa la sentencia DIM, se destruye cualquier tabla que existiese ya con el mismo nombre, y se crea una nueva tabla de forma que, inicialmente, tiene todos sus elementos iguales a Ø.

Además de poder definir tablas que se asocien a filas de variables (vectores o tablas de una dimensión), podrá definir tablas organizadas como cuadrículas de variables (matrices). Por ejemplo:

```
DIM a(1Ø,1Ø)
```

define un conjunto de variables dispuestas en 1Ø filas y 1Ø columnas. Un elemento particular de esta tabla se designará como a(i,j) y los subíndices seleccionan la fila y la columna respectivamente.

El concepto de tablas de dos dimensiones puede extenderse a tres, cuatro..., hasta 255 dimensiones. Es difícil imaginarse la disposición de las variables en tablas de más de dos dimensiones; pero se definen y utilizan de la misma forma que las de una y dos dimensiones. Así:

```
DIM a(1Ø,2Ø,5)
```

es una tabla de tres dimensiones, y un elemento cualquiera sería a(2,1,4). No se suelen emplear mucho las tablas de más de dos dimensiones. Esto es bueno, porque las tablas, como ya dijimos, gastan mucha memoria.

También puede definir tablas de variables de cadenas, y se pueden utilizar para guardar y manipular listas de palabras. Hay, no obstante, una complicación adicional, puesto que, en algunos sentidos, una cadena es ya una tabla, unidimensional, de caracteres.

Si usted define una tabla de caracteres de una dimensión, lo que en realidad obtiene es una variable de cadena de longitud fija:

```
DIM s$(1Ø)
```


es una variable de cadena que siempre almacenará 10 caracteres, sean los que sean (incluso si algunos son espacios en blanco). Para ver la diferencia entre la variable r$ y la tabla de cadena s$, pruebe este programa:

```
Ø DIM s$(1Ø)
2Ø LET s$="abc"
3Ø LET r$="abc"
4Ø PRINT s$, "x"; r$, "x"
```

verá como s$ siempre es de longitud 1Ø, sin importar lo que usted almacene en ella. Hay que darse cuenta de que en las tablas de cadena se pueden guardar valores en más de un elemento a la vez. A pesar de todo, también pueden almacenarse caracteres únicos en cada elemento. Las líneas siguientes almacenarán "z" en s$(5):

```
5Ø LET s$(5)="z"
6Ø PRINT s$
```

Tal vez se percate de la similitud entre esto y el troceado de cadenas de la sección anterior.

Puede manejar listas de palabras usando tablas de cadena de dos dimensiones. Por ejemplo, el programa para listar números en orden inverso al que se leen, puede utilizarse para invertir una lista de palabras:

```
1Ø DIM a$(5,1Ø)
2Ø FOR i=1 TO 5
3Ø INPUT a$(i)
4Ø NEXT i
5Ø FOR i=5 TO 1 STEP -1
6Ø PRINT a$(i)
7Ø NEXT i
```

Observe que si quita el último índice en una tabla de cadena de dos (o más) dimensiones, dicha tabla estará funcionando como un conjunto de cadenas:

a$(5,2) es un carácter único

pero

a$(5) es una cadena de 1Ø caracteres

Incluso se puede usar el troceado para extraer subcadenas y:

a$(5,1 TO 2)

sería una subcadena que comienza con el primer carácter y termina con el segundo carácter de la cadena a$(5).

El BASIC de Sinclair permite un uso muy flexible de las tablas y las cadenas. Eso puede dificultar, al principio, su comprensión; pero una vez que se acostumbre a ellas, verá que son muy poderosas.

Un juego de palabras

Como un caso del uso de tablas de cadena, consideremos el problema de escribir un programa para jugar al "juego del ahorcado". Como el computador no puede "pensar" una lista de palabras, es necesario pedir a alguien que introduzca las palabras, para que usted las adivine.

Una vez que se introduce la lista de palabras, el jugador debe intentar descubrir cada palabra letra por letra. Cada letra que se introduzca, se compara con cada letra de la palabra "oculta". Si ésta contiene la letra, en esta palabra el programa sustituye la letra con un blanco, para saber que ya ha sido adivinada por el jugador. Cuando se han descubierto todas las letras, el programa pasa a la palabra siguiente de la lista que se introdujo o, si ya no hay más, termina.

Después de esta ligera descripción, debería intentar escribir su propio programa del "ahorcado", antes de ver el que ahora le presentamos:

```
1Ø DIM w$(5,1Ø)
2Ø FOR i=1 TO 5
3Ø INPUT "palabra= "; w$(i)
4Ø NEXT i
5Ø FOR i=1 TO 5
6Ø LET g=Ø
7Ø LET t$=w$(i)
```



```
8Ø LET g=g+1
9Ø INPUT "letra= "; a$
1ØØ LET f=Ø
11Ø FOR j=1 TO 1Ø
12Ø IF a$(1)=w$(i,j) THEN PRINT "SI!-"; a$(1): LET
     w$(i,j)=" "
13Ø IF w$(i,j)<>"" THEN LET f=1
14Ø NEXT j
15Ø IF f=1 THEN GOTO 8Ø
16Ø PRINT "lo logro en"; g; "!!"
17Ø PRINT "la palabra era:"; t$
18Ø NEXT i
19Ø PRINT "fin del juego"
```

La línea 1Ø define la tabla w$, de forma que pueda guardar cinco palabras de hasta diez letras cada una. Las líneas de 2Ø a 4Ø son para introducir las palabras. El bucle FOR, que comienza en la línea 5Ø y termina en 18Ø, repite cinco veces la parte de "adivinación" del programa, una por cada palabra. La variable t$ se emplea en la línea 7Ø para guardar la palabra hasta el final del juego, de manera que el elemento correspondiente de la tabla pueda modificarse borrando letras con espacios, cuando éstas se van acertando.

Los intentos se introducen en la línea 9Ø. Cada letra de la palabra se compara con el intento actual (cada intento en una letra) por medio del bucle FOR de las líneas 11Ø-14Ø. La variable "f" se emplea para saber si en la palabra aún hay letras que deben ser descubiertas.

Inicialización de variables: DATA y RESTORE

Es frecuente que se necesite un conjunto típico de valores en un conjunto de variables o en una tabla, para que un programa funcione correctamente. Por ejemplo, si queremos imprimir el número de días en un mes concreto, podríamos crear una tabla de doce elementos y almacenar la respuesta en cada uno, es decir: el número de días de enero se guardará en el primer elemento, el número de días de febrero en el segundo, y así sucesivamente. Esta sería una idea útil y sencilla; pero, ¿cómo daríamos inicialmente el valor correcto a cada elemento de la tabla? Podría usar un bucle FOR y una sentencia INPUT para

introducir correctamente los valores. O también podría escribir doce sentencias LET. Para que estas cosas resulten ligeramente más fáciles, BASIC proporciona la capacidad de almacenar datos dentro de un programa, de tal manera que pueden ser transferidos a cualquier variable que desee. Los datos están almacenados en una sentencia DATA, que se compone de la palabra DATA seguida por una lista de valores separados por comas. Así, en el ejemplo propuesto de los días de cada mes, escribiríamos:

```
1Ø DATA 31,28,31,3Ø,31,3Ø,31,31,3Ø,31,3Ø,31
```

Para transferir estos valores a las variables, se utiliza la sentencia READ. Una sentencia READ consta simplemente de la palabra clave READ, seguida de una lista de variables separadas por comas. Cada vez que se encuentra una sentencia READ, se transfieren los valores de los datos a las variables de la lista, de forma que a cada variable se le transfiere un dato. Esta acción se ve más fácilmente imaginando un puntero colocado, inicialmente, sobre el primer dato de los que hay en la sentencia DATA. Cada vez que READ transfiere un dato a una variable de su lista, el puntero se mueve al siguiente valor de la lista DATA. Siempre que se ejecuta la sentencia READ, se transfiere un dato, empezando por el siguiente al último que se leyó, de los de la sentencia DATA, en ocasiones anteriores, en que se ejecutó READ. Por ello, una tabla que tenga los doce valores de la sentencia DATA que vimos como ejemplo, puede rellenarse usando:

```
2Ø DIM m(12)
3Ø FOR i=1 TO 12
4Ø READ m(i)
5Ø NEXT i
```

En un programa puede tener tantas sentencias DATA como desee, en cualquier lugar del programa, y todas serán tratadas como si todos los datos que llevan, estuviesen contenidos en una gran sentencia DATA. Si el puntero se mueve más allá del final de una DATA, continuará apuntando ahora al primer valor de la siguiente sentencia DATA. Sin embargo, si ya no quedan más sentencias de esa clase, obtendrá un mensaje de error.

No hay restricciones sobre la clase de datos que se pueden

guardar en una sentencia DATA; o sea, puede tener en ellas números o letras. Unicamente, deberá cuidar que los datos leídos al ejecutar READ correspondan a variables del tipo adecuado para almacenarlos: deben leerse cadenas en variables de cadena, y números en variables numéricas. Las cadenas colocadas en listas DATA deben estar encerradas entre comillas, pues, en caso contrario, no sería posible saber dónde termina un dato y dónde comienza el siguiente. Como ejemplo del empleo de este mecanismo, vamos a considerar:

```
1Ø DATA "Ene", "Feb","Mar","Abr","May","Jun",
   "Jul","Ags","Sep","Oct","Nov","Dic"
2Ø DIM m$(12)
3Ø FOR i= TO 12
4Ø READ m$(i)
5Ø NEXT i
```

y compárelo con el ejemplo que vimos antes.

Algunas veces, especialmente en los juegos, para lograr un determinado fin sería mucho más sencillo poder alterar la posición del puntero imaginario de las sentencias DATA. Esto puede efectuarse con la sentencia RESTORE. Si introduce en el programa RESTORE, seguido por un número de línea, la siguiente sentencia READ, empezará tomando sus datos del principio de la sentencia DATA más cercana de las que van debajo de la sentencia, cuyo número especificó (si no existe ninguna, nuevamente obtendrá un mensaje de error). Si usa RESTORE únicamente, sin número de línea, el puntero se coloca al principio de la primera sentencia DATA que se encuentre en el programa. Compruébelo con el programa:

```
1Ø DATA 1,2,3,4
2Ø READ a,b,c,d
3Ø PRINT a,b,c,d
4Ø RESTORE
5Ø GOTO 2Ø
```

Grabación de datos en cinta

Ya vimos que era posible almacenar (SAVE) y recuperar (LOAD) programas grabados en un magnetofón. Parece lógico

ampliar esta capacidad para que se pueda hacer lo mismo con los datos alamacenados en las variables: grabar y recuperar datos de una cinta. El Spectrum sólo permite grabar y recuperar tablas; pero eso no es una desventaja, pues cualquier variable que usted quisiera almacenar en el cassette, podría ser transferida primero a un elemento libre de una tabla, y grabar después esta última en cinta. Todo lo que tiene que hacer para almacenar una tabla en la cinta es proceder como lo haría si fuese a guardar un programa (en cuanto a conexiones y demás operaciones, que ya vimos), y sólo debe cambiar la sentencia:

SAVE "nombre de fichero" DATA "nombre de tabla" ()

donde "nombre de fichero" es una constante de cadena o una variable de cadena, que contenga el nombre del fichero de datos, y "nombre de tabla" es el nombre de la tabla (de cadena o numérica) que se va a grabar, sin olvidar añadir los paréntesis vacíos al final. Del mismo modo, para leer una tabla previamente almacenada use:

LOAD "nombre de fichero" DATA "nombre de tabla" ()

Este comando buscará en la cinta hasta que se encuentre una tabla, grabada previamente, con el nombre del fichero correcto. Cualquier tabla que existiese con el mismo nombre se borra, y la versión grabada se lee de la cinta, quedando lista para emplearla. Las sentencias SAVE y LOAD pueden ser usadas perfectamente, desde dentro de un programa, como líneas de éste.

Además, puede escribir:

VERIFY "nombre de fichero" DATA "nombre de tabla" ()

para comprobar que los datos se han registrado sin error en la cinta como hacíamos en los programas.

Como muestra, ahí va el siguiente programa:

```
10 DIM a(10)
20 FOR i=1 TO 10
30 LET a(i)=i
40 NEXT i
50 PRINT "rebobine la cinta"
60 SAVE "test" DATA a()
```



```
70 PRINT "rebobine la cinta y pulse la tecla PLAY"
80 LOAD "test" DATA a()
90 PRINT "datos correctamente cargados"
```

Todo, salvo acordarse de pulsar los controles correctos del grabador, es tan fácil como eso.

Capítulo 6
Funciones y subrutinas

En este punto de su aprendizaje de BASIC, debería ser capaz de ver que la expresión es la herramienta principal de los programas para **cambiar el valor de los datos.** Sin expresiones (aritméticas, de cadena o condicionales), BASIC se reduciría a un lenguaje para mover valores de un lugar a otro de la memoria. Los nuevos resultados sólo se producen gracias al empleo de expresiones, que pueden cambiar y combinar valores. Para que las expresiones sean aún más útiles, BASIC proporciona una extensa gama de operaciones que pueden ser agrupadas bajo el nombre de **funciones.** También puede usted ampliar el conjunto de funciones creando sus propias: **«funciones definidas por el usuario».** Esta creación de nuevas operaciones puede llevarse aún más lejos gracias a la sentencias GOSUB y RETURN, que permiten construir **unidades funcionales o subrutinas,** compuestas por un conjunto de sentencias.

En la primera parte de este capítulo, daremos una idea general de lo que son las funciones, para pasar después a examinar algunas de las más comunes, disponibles en el Spectrum. Las secciones que tratan de funciones concretas, puede leerlas rápidamente, e incluso saltarlas, hasta que necesite emplearlas, o hasta que las vea aparecer en uno de los ejemplos. Sin embargo, no pase por alto la sección sobre **funciones especiales,** puesto que aquellas son bastante importantes.

Concepto de función

Antes de tratar el modo en que el Spectrum maneja funciones, veremos primero las funciones en general. Puede que usted esté ya familiarizado con el concepto de función, si ha estudiado matemáticas. Por ejemplo, seno(x) es una función. No obstante, la idea de función realmente no tiene nada que ver con matemá-

ticas superiores. Al nivel más elemental, una función es una operación efectuada sobre algún dato, que produce un **único valor** como resultado. Así, encontrar el mayor de dos números es una operación sobre datos que devuelve un valor único (el máximo de 3 y 42 es 42, el máximo de 2 y 2 es 2, etc.), por lo que el **máximo** es una función.

El Spectrum no tiene una función máximo, pero, a pesar de todo, es útil tomarla como ejemplo, pues es fácil de comprender y, como veremos más avanzado el capítulo, es sencillo remediar su falta y programar una función máximo propia.

La forma normal de escribir una función comprende su nombre y los valores sobre los que opera, encerrados entre paréntesis. En el caso de encontrar el máximo de dos números, podemos tomar "max" como nombre significativo, y entonces nuestros dos ejemplos anteriores se escriben:

máx (3,42)
y
máx (2,2)

Siguiendo la notación usual de BASIC de poder escribir expresiones en cualquier parte donde puedan usarse constantes y variables, también sería correcto poner:

max(cuenta+3,total*20)

Los valores de los datos que siguen al nombre de la función se llaman **parámetros.** Es posible que una función tenga cualquier número de parámetros; pero todas las funciones que proporciona el Spectrum tienen uno solo.

Las funciones pueden aparecer en las expresiones como si fuesen variables o constantes:

LET resultado=máx(3,3,4,2)

Eso serviría para calcular el valor de nuestra función "máx", y almacenaría aquel (4) en "resultado" (Recuerde que el Spectrum no tiene la función "max", y, por ello, no puede probar este ejemplo.) Ahora puede entender por qué es tan importante que una función dé como resultado un valor único. Si no fuese así; ¿cuál de los diferentes valores que podría dar usaríamos para evaluar el resto de la expresión donde apareciese la función? o, ¿qué valor de los posibles almacenaríamos en la variable "resultado" del ejemplo? Como queremos escribir funciones dentro de las expresiones, deberán dar un único valor como resultado.

En ciertas ocasiones, es posible convertir en función algo que no lo es, simplemente eligiendo uno de los posibles valores que puede devolver. Por ejemplo, el Spectrum tiene la función SQR, que da como resultado la raíz cuadrada de un número. Si se pregunta que cuál es la raíz cuadrada de cuatro, la respuesta será, obviamente, dos: dos por dos da cuatro; pero también es demasiado fácil olvidar que: menos dos por menos da cuatro y, en consecuencia, menos dos es otra raíz cuadrada de cuatro (no olvide la regla del colegio de que "menos por menos, da más"). La objeción de que la raíz cuadrada no es, entonces, una función, puede ser fácilmente superada decidiendo simplemente, que SQR será una función que devuelve la **raíz cuadrada positiva** de un número.

Las funciones del Spectrum

El Spectrum tiene una gran variedad de funciones, y algunas de ellas están tan especializadas, que es mejor tratarlas detalladamente en otros capítulos. No obstante, hay un **grupo fundamental** de funciones que usted esperaría encontrar en cualquier BASIC, y éstas serán las que expliquemos en este capítulo. Se puede encontrar una lista completa (con una breve explicación) de todas las funciones del Spectrum en el Apéndice C del **Manual del Spectrum.**

Las **funciones fundamentales** se pueden dividir en tres grupos: las funciones aritméticas tales como SQR y ABS, las funciones trigonométricas como SIN y COS, y las funciones de cadena como LEN y CHR$. Además, existe una serie de funciones "inclasificables", pero muy importantes, tales como RND. Todas las funciones se introducen pulsando una sola tecla, y el uso de paréntesis para encerrar los parámetros es opcional. Lo más interesante es tener una idea de las funciones disponibles. Por ello, se recomienda una breve lectura de la descripción de cada función que aparece en la lista que viene después. Sin embargo, es difícil apreciar los detalles concretos del empleo de funciones, hasta que realmente tenga que usarlas. Si desea ver el efecto directo de cualquier función, puede emplear el programa:

```
1Ø INPUT x
2Ø PRINT SIN x
3Ø GOTO 1Ø
```


pero cambiando la sentencia PRINT SIN x, por la función que quiera comprobar en cada caso.

FUNCIONES ARITMETICAS

— ABS - VALOR ABSOLUTO DE UN NÚMERO.

El valor absoluto de un número se obtiene ignorando su signo, y tratándolo como si fuese positivo. O sea: ABS(−2)=2.

— EXP - FUNCIÓN EXPONENCIAL.

EXP(x) se calcula elevando el número "e", que vale 2.718281, a la potencia "x".

Esto es, EXP(x) es lo mismo que e^x.

Es difícil explicar por qué es tan importante esta función, pero está presente en todas las áreas de las matemáticas (vea también la función LN).

Tenga cuidado cuando utilice EXP(x), puesto que crece muy rápidamente, y los valores que devuelve son muy grandes, incluso para valores pequeños de "x". El máximo valor que puede tomar "x" en el Spectrum, para que éste pueda calcularlo correctamente, es 88. Valores mayores de "x", harán que se produzca el error: "Number too big" (número demasiado grande).

— INT - PARTE ENTERA DE UN NÚMERO.

La función INT es probablemente la más simple y la más utilizada en todas las funciones aritméticas. Su efecto es eliminar la parte fraccionaria de un número, dando como resultado un **número entero,** y además haciendo un redondeo por defecto. En los números positivos el resultado se obtiene eliminando simplemente la parte decimal. Ejemplo INT(3.21)=3; pero en los números negativos la cosa es un poco más compléja. Parece extraño redondear un número negativo por defecto: INT(−4.7)=−5, pero eso es consecuencia de que −4>−5 (o sea, −5 es más pequeño que −4).

— LN - LOGARITMO NEPERIANO (O NATURAL) DE UN NÚMERO.

El logaritmo neperiano de un número es la potencia a la que hay que elevar el número "e" para obtener el número de partida. Mucha gente conoce una clase de logaritmo ligeramente distinta: el logaritmo decimal (en base 10), que es el que aparece en muchas de las tablas de logaritmos. La definición de este último

es: logaritmo decimal de un número es el número al que hay que elevar 10 para obtener el número de partida. Algunas versiones de BASIC incluyen una función LOG, que calcula estos logaritmos decimales. El Spectrum carece de ella, pero no es ninguna desventaja, puesto que el logaritmo decimal se puede calcular como LN(x)/LN(10). Como LN(x) es un número al que hay que elevar "e" para obtener x, observe que EXP(LN(x))=x.

— PI - (π)

Esta es una función **extraña** que no tiene parámetros, y siempre devuelve el mismo valor: el del número $\pi=3.14159265$. A pesar de eso, es muy útil. Como todo el mundo sabe, el área de un círculo de radio r, viene dado por πr^2, y eso se pondría en BASIC como:

```
1Ø LET area=PI*r*r
```

— SGN - EL SIGNO DE UN NÚMERO.

El signo de un número es +1, si éste es positivo, y −1, si es negativo. Así, SGN (−3.25)=−1 y SGN (222)=+1.

— SQR - RAÍZ CUADRADA.

La raíz cuadrada de un número es un valor tal que, multiplicado por sí mismo, da el número original.

SQR(x)*SQR(x) es igual a x

Vea que los números negativos no tienen raíz cuadrada, porque si se multiplica cualquier número (aunque sea negativo) por sí mismo, siempre se obtiene un resultado positivo. Si intenta calcular la raíz cuadrada de un número negativo, se producirá un error que hará aparecer el mensaje muy preciso: "Invalid argument" (argumento inválido).

FUNCIONES TRIGONOMETRICAS

Los ejemplos más conocidos de funciones son, probablemente, las funciones trigonométricas. Va más allá del alcance de este libro, entrar en detalles de teoría de la trigonometría, y, de cualquier modo, la necesidad de usar estas funciones surge siempre en problemas muy específicos. No obstante, hay una aplicación de estas funciones que es muy importante para todos los

usuarios de computadores interesados en los gráficos: el dibujo de círculos. Si desea saber más sobre este tema, vea el capítulo 10 del Manual del Spectrum.

Los poseedores del Spectrum son particularmente afortunados, pues el BASIC de la máquina incluye un comando que dibuja círculos directamente. Este tema se trata en el capítulo nueve, y, gracias a dicho comando, no se necesita entrar en el empleo de senos, cosenos y tangentes para dibujar círculos.

Si necesita emplear funciones trigonométricas, es importante saber que el Spectrum no mide los ángulos en grados, sino en radianes (los Radianes como medidas de ángulos se verán también en el capítulo 9, en el tema de dibujo de arcos de círculo en la pantalla). Si desea convertir un ángulo medido en grados a radianes, puede usar:

```
radianes=grados*PI/18Ø
```

y para la conversión de radianes a grados, utilice:

```
grados=radianes*18Ø/PI
```

Las tres funciones trigonométricas disponibles en el Spectrum son:

SIN - seno de un ángulo medido en radianes
COS - coseno de un ángulo medido en radianes
TAN - tangente de un ángulo medido en radianes

Además, están disponibles las funciones inversas de cada una de éstas:

— ASN - ARCO SENO

El arco-seno de un número es el ángulo, medido en radianes, cuyo seno (SIN) es igual a dicho número. O sea: x=SIN (ASN(x)). Como SIN (x) siempre da un resultado comprendido entre −1 y +1, ASN(x) sólo puede emplear valores de x en ese intervalo.

— ACS - ARCO COSENO.

El arco-coseno de un número es el ángulo, medido en radianes, cuyo coseno (COS) es igual a ese número. Por tanto, x=COS(ACS(x)). Por iguales motivos, los valores permitidos

para x son entre −1 y +1, pues nuevamente COS(x) está limitado por esos valores.

— ATN - ARCO TANGENTE.
El arco-tangente de un número es el ángulo, en radianes, cuya tangente (TAN) es igual a ese número: x=TAN(ATN(x)).

FUNCIONES DE CADENA

Ya vimos algunas de ellas en el capítulo 5.

— CHR$ - FUNCIÓN CARÁCTER.
CHR$ n dará el **carácter** que ocupa la n-sima posición en la lista de todos los caracteres del Spectrum. Observe que todas las palabras clave, como LET, cuentan como un carácter cada una, desde el punto de vista del Spectrum. Por lo tanto, CHR$ puede devolver una cadena de más de una letra. CHR$ puede devolver todos los caracteres que maneja el Spectrum, incluso aunque algunos de ellos no pueden ser representados en pantalla. Así, si teclea CHR$(8), o cualquier número hasta 33 inclusive, todo lo que verá en la televisión será la pantalla en blanco.

— CODE - CÓDIGO DE UN CARÁCTER
La función CODE es la inversa de CHR$, ya que su resultado es la posición en la lista de caracteres de cualquier carácter usado como parámetro de ella. Ejemplo: CODE "A" es 65 y CHR$(65) es "A". Si aplicamos CODE a una cadena de más de un carácter, el código que nos devolverá la función será el que corresponde al primer carácter de la cadena. Si la cadena es vacía (sin ningún carácter), el código devuelto es cero.

— LEN - LONGITUD DE UNA CADENA.
Devuelve la longitud de cualquier cadena: LEN "computador" dará diez, y LEN "" (cadena vacía), será cero.

— STR$
Esta función es útil en aplicaciones avanzadas. Convierte cualquier número (o el resultado de una expresión), en la cadena de caracteres que se verían en la pantalla, si el número (o el resultado de la expresión) fuese escrito en dicha pantalla. La

función STR$ proporciona una conexión entre números y cadenas. Por ejemplo: STR$(31)+"de Julio" nos daría como resultado la cadena "31 de Julio".

— VAL - EVALUAR EXPRESIÓN ARITMÉTICA.
Esta es la inversa de STR$. VAL convierte una cadena en un número. La cadena puede ser cualquier expresión aritmética correcta, y el número resultante será el valor de la expresión. Por ejemplo: VAL "3+3*6" sería 21.

FUNCIONES ESPECIALES

Hay dos funciones, RND e INKEY$, que son tan útiles que parece una buena idea tratarlas aparte y con alguna extensión.
— RND devuelve un número y, por ello, podríamos haberla incluido entre las funciones aritméticas.
— INKEY$ devuelve un carácter, y podríamos haberla incluido como función de cadena.

RND

Esta es una función sin parámetros que devuelve un número en el intervalo de 0 a 1, excluido el propio 1, que puede ser tratado como si fuese aleatorio. La afirmación de que un computador puede dar números aleatorios, parece siempre una contradicción, y en cierta forma lo es. La confusión se debe al uso de la palabra **«aleatorio».** Si utiliza el computador para jugar ciertos juegos, todo lo que necesita es una secuencia de números que no puedan preverse por parte de ninguno que juegue con la máquina. En otras palabras, para la mayoría de los propósitos, se puede decir que una lista de números es aleatoria cuando no se puede detectar una relación entre ellos, ni tampoco una forma determinada de producirse. Si ejecuta el programa:

```
1Ø PRINT RND
2Ø GOTO 1Ø
```

verá una lista de números que no muestran ninguna relación predeterminada (de hecho sí que existe tal relación, pero es tan compleja que necesita un Spectrum para generarla). Esta clase de aleatoriedad es conocida con el nombre más técnico de

pseudoaleatoriedad, y la función RND es un generador de números pseudoaleatorios. Los números producidos están **uniformemente distribuidos** en el intervalo de Ø a 1, excluido este último ([Ø,1) en matemáticas). Es decir: cualquiera de los números tiene igual probabilidad de obtenerse que los demás y no podríamos encontrar una relación visible que ayudase a predecir cuál será el siguiente número que RND generará.

El problema principal de RND es que escasas veces necesitaremos números aleatorios en el rango que los produce. Normalmente, querremos que el programa haga diferentes cosas aleatoriamente. La mejor forma de hacerlo es cambiar el número generado por RND en un número entero entre 1 y n, donde n es el número de cosas diferentes entre las que se puede elegir "al azar". La fórmula:

```
INT(RND*n)+1
```

es la apropiada para obtener un entero pseudoaleatorio en el intervalo de 1 a n. Por ejemplo, si desea un programa que simule el lanzamiento de un dado, elegirá seis como valor para n, y:

```
1Ø PRINT INT(RND*6)+1
2Ø GOTO 1Ø
```

imprimirá números del 1 al 6 aproximadamente con la misma frecuencia, y de forma que será impredecible el siguiente número impreso (al menos, no es nada evidente). El tema de cómo usar los números aleatorios en los programas, es demasiado amplio para cubrirlo completo en este libro, pero encontraremos algunos ejemplos de ello en capítulos posteriores.

La función RND es especial, además, porque está asociada con otro comando BASIC: RANDOMIZE (aleatorizar). La lista de números que produce RND no es ilimitada. Después de 65536 valores, se repite otra vez. Cada vez que usted conecta el Spectrum, la lista comienza en el mismo lugar. Si quiere comprobar esto, primero apague el computador y enciéndalo de nuevo; después introduzca el programa que imprime números aleatorios hasta llenar la pantalla (el que vimos antes). Cuántas veces haga esto, apagar y encender el Spectrum, volverá a obtener la misma lista de números producidos por RND, y en la misma secuencia. Obviamente, esto no es una buena idea si

quiere tener juegos, puesto que usted podría aprenderse la secuencia producida cuando se enciende el Spectrum de nuevo.

Por otra parte, realmente podría querer generar la misma secuencia cada vez, y tener que apagar y encender el computador en cada ejecución crearía dificultades. A fin de evitar esto, se puede usar RANDOMIZE (introducido pulsando la tecla marcada con RAND) para empezar la secuencia de nuevo. Este comando, en la forma RANDOMIZE n, empezará la secuencia desde el n-simo número aleatorio de la lista que el Spectrum tiene de ellos (que ya sabemos es fija y de 65.536 valores). Por ejemplo, si introduce:

```
1Ø RANDOMIZE 3Ø
2Ø PRINT RND
3Ø GOTO 2Ø
```

logrará la misma secuencia de números cada vez que lo ejecute, sin tener que apagar y encender de nuevo la máquina. Si emplea RANDOMIZE Ø ó RANDOMIZE sin ningún número de comienzo, el Spectrum empleará un número relacionado con el tiempo transcurrido desde que fue conectado para comenzar la secuencia. Pruebe:

```
1Ø RANDOMIZE Ø
2Ø PRINT RND
3Ø GOTO 2Ø
```

que escribe el primer número de la secuencia una y otra vez. Observe que los números que se obtienen se incrementan lentamente, contando el tiempo que lleva encendido el Spectrum. La secuencia "más aleatoria" que puede lograrse combinando RND y RANDOMIZE, puede verse ejecutando:

```
1Ø RANDOMIZE Ø
2Ø PRINT RND
3Ø GOTO 2Ø
```

INKEY$

La función INKEY$ está íntimamente relacionada con INPUT, en el sentido de que puede usarse para **leer** un carácter

único del teclado. La diferencia es que INPUT A$ espera a que se introduzca algo por el teclado y se pulse ENTER, pero INKEY$ no espera. Si prueba el programa siguiente:

```
1Ø INPUT A$
2Ø IF A$<>"" THEN PRINT A$
3Ø GOTO 1Ø
```

tendrá que pulsar ENTER para ver algo en la pantalla (para obtener este programa, necesitará borrar las comillas de la izquierda del cursor, y pulsar STOP; después, como siempre, ENTER). Sin embargo, si cambia la línea 1Ø por ésta:

```
1Ø LET A$=INKEY$
```

El carácter que corresponda a la tecla que usted pulse, aparecerá directamente en la pantalla (sin usar ENTER). Puede intentar ver qué ocurre pulsando más de una tecla a la vez, o pulsar SHIFT y las otras teclas. Hay que resaltar otra diferencia entre INPUT e INKEY$: resulta que la segunda no imprime nada en la parte inferior de la pantalla cuando se pulsa una tecla.

Cada vez que el Spectrum encuentra la función INKEY$, examina inmediatamente el teclado. Si ya hay una tecla pulsada, la función devuelve el carácter correspondiente a esa tecla. Pero si no hay ninguna tecla pulsada, la función devolverá la cadena vacía. Pase lo que pase, INKEY$ nunca esperará a que se pulse ninguna tecla.

El principal uso de esta función son los juegos, donde las cuatro flechas de desplazamiento del cursor se emplean para mover algo en la pantalla. Por ejemplo:

```
1Ø LET a$=INKEY$
2Ø IF a$="" THEN GOTO 1Ø
3Ø IF a$="5" THEN PRINT "IZQUIERDA"
4Ø IF a$="6" THEN PRINT "ABAJO"
5Ø IF a$="7" THEN PRINT "ARRIBA"
6Ø IF a$="8" THEN PRINT "DERECHA"
7Ø GOTO 1Ø
```

La línea 1Ø toma el carácter correspondiente a la tecla que se pulse, si hay alguna pulsada. La línea 2Ø comprueba si a$ es la

cadena vacía, o sea, si no hay ninguna tecla pulsada, y si es así, devuelve el control de nuevo a la línea 1Ø. Entonces, el bucle que forman las líneas 1Ø y 2Ø sirve para esperar a que se pulse alguna tecla. Después, las líneas 3Ø a 6Ø investigan la tecla que se pulsó y dan el mensaje adecuado, siempre que lo que se haya pulsado sea una de las flechas del cursor. La línea 7Ø repite el programa completo. Como puede ver, si se pulsan teclas diferentes a las flechas, el bucle 1Ø-2Ø esperará hasta detectar una tecla apretada, pero no se imprimirá nada en la pantalla. En el capítulo 9, se verá un ejemplo donde la misma clase de programa se emplea para dirigir un punto por la pantalla.

Funciones definidas por el usuario: DEF FN y FN

En nuestra introducción general a las funciones, discutimos la idea de una función para encontrar el máximo de dos números, señalando que, aunque el Spectrum tiene una amplia gama de funciones, dicha función (máx) no se cuenta entre ellas. El BASIC del Spectrum permite definir nuevas funciones; pero hay una serie de ideas algo complicadas relacionadas con esas definiciones.

Se puede definir una nueva función en términos de una expresión. Por ejemplo, el Spectrum carece de la función "cuadrado de un número", que calcula el resultado de elevar un número al cuadrado. Los nombres de las funciones definidas por el usuario son "FN", seguido de una única letra. Sabiendo eso, puede llenar el vacío de la función cuadrado con:

```
DEF FN c(x)=x*x
```

(DEF FN se obtiene con una sola tecla. La encontrará en la esquina izquierda de la fila superior del teclado, bajo la tecla 1.) La palabra DEF se emplea para indicar qué es una definición de función. El significado de esta función es bastante claro (cada vez que el Spectrum ve una función llamada "FNc" eleva al cuadrado el parámetro encerrado entre paréntesis). Ejemplo:

```
LET a=FNc(2)
```

asignaría a la variable "a" el valor 4 (de nuevo, FN debe ser

introducido con una sola tecla. Está bajo la tecla "2"). Todo esto está muy bien. Pero, ¿qué ocurriría con el programa?:

```
1Ø DEF FNc(x)=x*x
2Ø LET x=3
3Ø LET z=4
4Ø LET a=FNc(z)
5Ø PRINT "x="; x; " z="; z; " a="; a
```

Antes de ejecutar el programa, intente averiguar qué valores darían "x", "y", "z" al imprimirlos. La respuesta correcta es que "x" sería 3, "z" sería 4, y "a" sería 16. Cualquier confusión posible será a causa de que "x" se usa en la definición de la función "FNc", y también en el programa. Una cosa a destacar es que los nombres que emplee en la definición de una función no tienen nada que ver con cualquier variable que luego pudiese usar en el programa. En este sentido, DEF FNc(a)=a*a, DEF FNc(z)=z*z, etc., definen todas la misma función.

En la definición de una función, el parámetro que aparece sólo dice lo que va a ocurrir con los parámetros reales, cuando la función se use (por esto, se suelen denominar **«parámetros ficticios»).**

Si ha captado estas ideas, tal vez le guste intentar descubrir cuál es el resultado del siguiente programa:

```
1Ø DEF FN t(i)=total+i
2Ø LET total=Ø
3Ø PRINT FN t(3Ø)
4Ø LET total=1Ø
5Ø PRINT FN t(3Ø)
```

En este caso, la variable "total", que se usa en la expresión de definición de la función, no es un **parámetro ficticio,** y, por lo tanto, se usará como una variable en el programa principal, es decir, "total" de la definición de la función es lo mismo que "total" en el resto del programa.

El concepto de parámetro ficticio será más fácil de entender después que haya reflexionado un poco sobre él.

Para resumir, las reglas para crear una función definida por el usuario son:

(1) Cualquier función que emplee debe ser definida mediante la sentencia DEF FN en alguna parte del programa (no necesariamente antes de que sea usada).

(2) Se permiten tanto funciones numéricas como de cadena. Las funciones numéricas que devuelven como resultado un número, tienen nombres de la forma "FN" seguido de una única letra. Las funciones de cadena, que dan como resultado una cadena, tienen nombres de la forma "FN" seguido de una única letra y el símbolo del dólar ($).

(3) Una definición de función puede incluir cualquier número de parámetros ficticios (o ninguno), que pueden representar parámetros numéricos o de cadena. Todos los parámetros ficticios tienen nombres de una sola letra, y los parámetros ficticios de cadena deben terminar con el signo del dólar. Los paréntesis deben usarse aunque la función no tenga parámetros.

(4) Cualquier expresión BASIC correcta puede emplearse en una definición de función.

Estas cuatro reglas parecen difíciles de recordar, pero todas se guían por el sentido común. Algunos ejemplos de funciones ayudarán a aclarar los conceptos:

```
DEF FN m$(a$,i,j)=a$(i TO i+j-1)
DEF FN l(x)=LN x/LN 1Ø
DEF FN d()=INT(RND*6)+1
```

La primera función, "m$", es una función de cadena con un argumento de cadena y dos argumentos numéricos. Extraerá "j" caracteres de la cadena a$, empezando en el i-ésimo. Ejemplo: m$ ("HOLA", 2,3) dará "OLA".

La segunda función "l" es una función de un único parámetro ficticio numérico que calcula el logaritmo decimal de z (vea la nota anterior sobre la función LN).

Fíjese en que la afirmación: "cualquier expresión BASIC correcta", que hace el punto (4), incluye expresiones formadas con otras funciones predefinidas en BASIC.

El ejemplo final no tiene parámetros, pero observe que, aún así, los paréntesis deben estar presentes en la definición. Esta devuelve como resultado un número aleatorio en el rango del 1 al 6 (vea los detalles de RND en la lista de funciones predefinidas que dimos antes). Usted también necesita poner "()" cuando emplea la función. Ejemplo: PRINT d().

Ahora podemos utilizar lo que hemos aprendido sobre fun-

ciones definidas por el usuario, para crear la función "máx", que anteriormente sirvió para presentar el concepto de función. Si se para a pensarlo, parece que no se podría usar una función utilizada por el usuario para definir "máx". Después de todo, ¿cómo podría definirse una expresión que elija entre dos números uno de ellos? Si comprendió bien la discusión sobre expresiones condicionales del capítulo 4, y recuerda que éstas daban "1", si eran ciertas, y Ø, si eran falsas, la respuesta al problema es fácil de entender:

```
1Ø DEF FN m(a,b)=a*(a>=b)+b*(a<b)
2Ø INPUT a
3Ø INPUT b
4Ø PRINT FN m(a,b)
5Ø GOTO 2Ø
```

Si ejecuta el programa verá que la línea 4Ø escribe siempre el mayor de los dos números que usted teclea como respuesta a las líneas 2Ø y 3Ø. La función en sí trabaja correctamente, porque sólo una de las condiciones "(a>=b)" y "(a<b)" puede ser cierta y, por ello, una dará siempre resultado "1", y la otra "Ø". Como cero por cualquier número da cero, se ve que realmente la función selecciona uno de los dos números.

Este ejemplo viene a reforzar, una vez más, el hecho de que en una definición de función se puede incluir cualquier expresión BASIC correcta.

Subrutinas: GOSUB y RETURN

La idea de crear nuevas operaciones definiendo funciones es muy poderosa, pero es posible que se encuentren limitaciones en ella, rápidamente. Algunas veces, sería una ventaja dar un nombre no sólo a una línea BASIC (en forma de función definida por el usuario), sino también a un conjunto completo de líneas. Esta es la idea que hay detrás de las **subrutinas.** Una subrutina no es nada más que un grupo de sentencias BASIC que pueden usarse tan a menudo como se desee, únicamente escribiendo su nombre (de igual forma que el nombre de una función definida por el usuario). Lo malo es que las subrutinas BASIC son muy limitadas. No se puede dar ni siquiera un nombre con una única letra a una subrutina. Para referirse a éstas hay que emplear el

número de su primera línea, y no se pueden incluir parámetros de ningún tipo. Incluso con esas restricciones, es aún bastante recomendable conocer y usar las subrutinas BASIC.

Si las líneas BASIC de que consta una subrutina comienzan en la línea "n", puede utilizar ésta con:

```
GOSUB n
```

Donde GOSUB quiere decir "ve a la subrutina" (GO to SUBbroutine, en inglés). Realmente, la acción de GOSUB se parece mucho a la de GOTO, pues ambas transfieren el control a la línea "n". La diferencia entre ambas sentencias es que el comando GOSUB hace que el Spectrum almacene el número de línea donde está el GOSUB en un área especial de memoria que se reserva para esos fines. Este número de línea almacenado lo utiliza la sentencia RETURN para transferir el control a la línea siguiente a la que ocupa GOSUB, cuando la subrutina ha terminado.

Por ejemplo, el efecto de GOSUB y RETURN en el flujo de control es:

```
        1Ø GOSUB 1ØØØ - - - - - - - -
 - - -> 2Ø PRINT a                  |
          . . .                     |
        resto del programa          |
          . . .                     |
          . . .                     |
        1ØØØ LET a=56  <- - - - - - -
 - - - - 1Ø1Ø RETURN
```

La línea 1Ø transfiere el control a la subrutina que comienza en 1ØØØ, que únicamente almacena en "a" el valor 56. La sentencia RETURN de la línea 1Ø1Ø finaliza la subrutina y, automáticamente, devuelve el control a la línea 2Ø (que sigue a la línea de GOSUB, la 1Ø).

En una subrutina se puede utilizar cualquier sentencia BASIC que sea válida en un programa. En particular, no hay nada que impida que una subrutina tenga un GOSUB en su interior y transfiera el control, a su vez, a otra subrutina. Si usted hace esto, el siguiente RETURN devolverá el control a la sentencia siguiente al GOSUB último que se ejecutó. En otras palabras: si una subrutina llama a otra, entonces RETURN se comportará

como sería de esperar, transfiriendo de nuevo el control al sitio donde cada subrutina fue llamada, es decir, un RETURN nunca olvida de donde vino.

El par GOSUB y RETURN es todo lo que existe en BASIC para manejar subrutinas. Todas las variables de una subrutina son las mismas que las del programa principal, no hay ninguna clase de parámetros. Como ya mencionamos antes, esto podría conducirle a pensar que las subrutinas no son muy útiles: ¡eso está lejos de ser cierto!

Uso de las subrutinas

Si lee usted el Manual del Spectrum (capítulo 5) llegará a la conclusión de que la forma más útil de emplear una subrutina es reemplazar un trozo de programa que sea necesario más de una vez. Por ejemplo, si en un programa largo necesita imprimir el mismo mensaje una vez tras otra, entonces es mejor colocar esa línea en una subrutina, y mandar el control a ésta cada vez que se necesite el mensaje. Aunque esta es una aplicación importante de las subrutinas, muy a menudo ocurre que es una buena idea reunir partes de un programa en subrutinas, aunque sólo se utilicen una vez durante toda la ejecución. La razón de esto es que los programas que emplean subrutinas son mucho más fáciles de entender, es más sencillo encontrar sus errores, y son más fáciles de modificar. Esta no es la clase de afirmación que puede demostrarse, puesto que el concepto de sencillez en este terreno es, obviamente, una cuestión de opiniones.

El empleo de subrutinas en la escritura de programas BASIC se ilustrará con los ejemplos del resto del libro. Si usted descubre un método de programar que le guste más, nadie podrá decir nada en contra suya.

Todo lo que puedo decir es que muchos programadores están de acuerdo en que las subrutinas son una buena cosa.

Capítulo 7
Gráficos

El atractivo principal de programar el Spectrum es emplear sus gráficos y sonido. Este capítulo comienza presentando la clase de gráficos que se pueden obtener utilizando sentencias PRINT: Los **gráficos de baja resolución.** El capítulo 8 añade el sonido, y el capítulo 9 trata del dibujo de imágenes de forma más detallada: los **gráficos de alta resolución.** No debería usted confundirse, creyendo que los gráficos de alta resolución son, de alguna forma, más útiles que los de baja resolución. En realidad, no son más avanzados unos que otros: sólo diferentes. Y es sorprendente cómo frecuentemente un programa funciona mejor y es más sencillo de escribir empleando gráficos de baja resolución.

Control de PRINT

Hasta el momento, hemos utilizado la sentencia PRINT para imprimir números y cadenas en la pantalla, ya sea uno en cada línea, o bien unos junto a otros en la misma línea. Para la mayoría de los programas esto sería suficiente, pero, a veces, se necesita controlar exactamente el sitio donde se imprimirá algo en la pantalla. En el capítulo 3, la sentencia PRINT se definió como:

PRINT "lista de impresión"

donde "lista de impresión" se dijo que era una lista de componentes, cada uno separado del otro por punto y coma. Se necesitan los puntos y comas, simplemente para saber dónde termina un componente de la lista y dónde comienza el siguiente. Por ejemplo, PRINT "total suma" intentaría escribir una variable única llamada "totalsuma" (recuerde que en los nom-

bres de las variables los blancos se ignoran), mientras que se escribiría PRINT total; suma, intentará imprimir dos variables, "total" y "suma", una junto a otra en una misma línea. El Spectrum permite realmente el uso de tres símbolos para separar componentes de la lista de impresión, y cada uno de ellos tiene un efecto diferente sobre la forma en que aparecerá la lista entera escrita en la pantalla.

El punto y coma ";" que hemos venido utilizando desde el capítulo 3, únicamente significa: imprimir el siguiente componente sin dejar ningún espacio. Si se emplea una coma (,) como separador, entonces la posición de impresión se mueve a la **zona** siguiente antes de escribir el componente que le sigue. La pantalla del Spectrum se divide en dos zonas de impresión: las primeras dieciséis posiciones de una línea y las dieciséis restantes de la misma línea. Por lo tanto, PRINT "a", "b" imprimirá "a" en la columna primera, y "b" en la columna número diecisiete. El tercer y último separador es el apóstrofo "'", que el Spectrum interpreta como una instrucción para empezar una nueva línea antes de imprimir el siguiente componente de una lista de impresión. Entonces:

PRINT "a'"'"b"

tendrá el mismo efecto que

PRINT "a"
PRINT "b"

Se puede intercalar más de un separador entre dos componentes de PRINT, sin que se produzcan "efectos perniciosos". Ejemplo: PRINT "a", "b" hará que el Spectrum escriba "a" y se mueva dos zonas de impresión antes de imprimir "b". Si lo piensa un poco, se dará cuenta de que eso significa que "a" aparecerá en la primera columna de una línea, y "b" en la primera columna de la línea siguiente. El caso más importante de empleo de varios separadores seguidos es poner varios apóstrofos, uno detrás de otro, para dejar varias líneas en blanco. Hay un caso especial que debe comentarse: si cualquiera de los separadores se coloca al final de una lista de componentes PRINT, queda automáticamente suprimido el cambio a una nueva línea de impresión que ocurría antes. Por ejemplo:

PRINT "a";
PRINT "b"

es lo mismo que PRINT "a"; "b"
y

PRINT "a"
PRINT "b"

Es lo mismo que escribir PRINT "a","b".

Si el separador en que acaba una lista es un apóstrofo, el cambio automático de línea queda suprimido, pero el propio apóstrofo provocará un cambio de línea, y, por tanto, no habrá diferencia en la pantalla con:

PRINT "a" y PRINT "a'"

Resumiendo:

separador	*efecto*
; (punto y coma)	imprimir siguiente componente sin dejar ningún espacio
, (coma)	pasar a la siguiente zona de impresión
' (apóstrofo)	empezar una nueva línea

Funciones de impresión: TAB y AT

El uso de los distintos separadores en las listas de impresión, ha incrementado bastante nuestro control sobre cómo se imprimen en la pantalla las cosas, pero aún no sabemos hacer que cierto componente de una lista PRINT aparezca en una determinada columna o en una determinada fila de la pantalla. Para conseguir eso, necesitamos algo más que los separadores.

Para controlar exactamente las posiciones donde se van a empezar a escribir los componentes de una sentencia PRINT, existen dos funciones: TAB y AT (ambas son funciones, y por ello deben obtenerse en el teclado como tales, y, además, son especiales, pues no devuelven ningún valor, y, en consecuencia, sólo se pueden usar dentro de una sentencia PRINT como parte de "lista de impresión").

TAB puede controlar la posición horizontal dentro de la línea actual de impresión, y AT es una función absolutamente general, que puede producir una salida en cualquier lugar de la pantalla.

La forma general de la función TAB es:

TAB "Expresión aritmética"

y sus efectos son mover la posición de impresión a la columna cuyo número es el resultado de la "expresión aritmética". O sea:

PRINT "a"; TAB 1Ø; "b"

escribirá "a" en la columna uno, y "b" en la columna 1Ø. Se pueden tener tantos TAB en una misma sentencia PRINT como se necesite:

PRINT "a"; TAB 1Ø; "b"; TAB 2Ø; "c"

Esta línea imprime "a" en la primera columna, "b" en la columna 1Ø, y "c" en la columna 2Ø.

Cuando se emplea la función TAB pueden surgir dos problemas: ¿qué ocurrirá si la posición actual de impresión en una línea ha sobrepasado la columna especificada en el siguiente TAB? y ¿qué pasa al especificar columnas inexistentes, como TAB 35 (sólo hay treinta y dos columnas)?

La respuesta a la primera pregunta es que el Spectrum se moverá a la columna cuyo número tiene TAB, pero en la **línea siguiente.** Puede verlo con la línea:

1Ø PRINT "a"; TAB 15; "b"; TAB 1Ø; "c"

que imprime "a" en la columna 1, "b" en la columna 15, y "c" en la columna 10 de la línea siguiente.

La respuesta a la segunda pregunta es que el Spectrum resta 32 del número de columna que usted especifica en TAB, hasta que obtenga un número entre 1 y 32. Por ejemplo, TAB 46 tiene el mismo efecto que TAB 14, puesto que 46-32 es igual a 14, y TAB 126 tiene igual efecto que TAB 3Ø, ya que: 126−32 son 94; 94−32 son 62 y 62−32, es igual a 30.

La función de impresión más poderosa es AT, porque puede dirigir la posición de impresión a una línea particular y una columna determinada. La forma general de AT es:

AT "expresión aritmética 1", "expresión aritmética 2"

El valor de la primera expresión aritmética es el número de la línea donde se escribirá el siguiente componente de una lista PRINT, y el valor de la segunda expresión es el número de la columna dentro de la línea dada. (Si los números de línea o columna obtenidos caen fuera de la pantalla, obtendremos un error: "Out of screen", fuera de pantalla, en inglés.

A diferencia de TAB, cuyas columnas de impresión se numeran de 1 a 32, con AT las columnas están numeradas empezando en la columna Ø, y terminando en la 31. De forma análoga, la numeración de las líneas va desde Ø, la línea superior de la pantalla, a 21, que corresponde a la parte baja de la pantalla. Por ejemplo:

PRINT AT 3,5;"a"

imprimirá "a" en la línea 3 (es decir, la cuarta línea, empezando por arriba) y la columna 5 (la sexta columna de impresión). Esto podría parecerle un poco complicado si está acostumbrado a las coordenadas x, y (cartesianas), en las que x es la distancia horizontal, é y la distancia vertical, puesto que para mover la posición de impresión á x, y, debería utilizar PRINT AT y,x.

Una diferencia interesante entre TAB y AT es que, si usa TAB para mover la posición de impresión a un determinado sitio, borrará todo lo que haya en la pantalla desde la posición actual de impresión hasta la nueva. Por ejemplo:

```
1Ø PRINT "***************"
2Ø PRINT AT Ø,Ø;
3Ø PRINT TAB (1Ø); "a"
```

No tiene importancia el número exacto de "*" que ponga en la línea 1Ø, éstas son sólo para observar el efecto de TAB. En esta demostración hay dos cosas interesantes. En primer lugar, puede usarse AT para mover la posición de impresión actual a cualquier otro sitio de la pantalla, sin tener que escribir por ello nada. En segundo lugar, la línea 1Ø muestra que la función TAB (1Ø) de la línea 3Ø borra todas los "*" desde el principio de la línea, hasta la columna 1Ø. Si quiere ver el efecto que provoca AT (no borra los "*", sino que imprime "a" en lugar del décimo *), cambie la línea 3Ø por esto:

3Ø PRINT AT Ø,1Ø;"a"

En la lista de impresión de una sentencia PRINT puede incluir tantos AT como desee. Cada uno mueve la posición de impresión al sitio indicado, antes de imprimir el siguiente componente de la lista. Aparte de eso, también puede mezclar AT y TAB, y todos los demás símbolos de control de la impresión, en una sentencia PRINT, de la forma que más le convenga. Por ejemplo:

```
1Ø PRINT AT 1Ø,5;"a";TAB(5);"a";AT 3,1Ø;"b""c"
```

será aceptada por el Spectrum sin ningún problema.

Como ejemplo del empleo de AT, intente escribir un programa que dibuje en la pantalla un cuadrado hecho con asteriscos. En caso de tener algún problema, una de las muchas respuestas posibles (siempre hay más de una forma de escribir un programa) es:

```
1Ø FOR i=Ø TO 1Ø
2Ø PRINT AT 5,i+1Ø;"*"
3Ø PRINT AT 15,i+1Ø;"*"
4Ø PRINT AT i+5,1Ø;"*"
5Ø PRINT AT i+5,2Ø;"*"
6Ø NEXT i
```

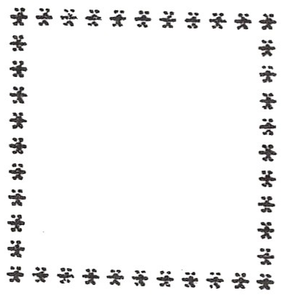


Fig. 7.1. Un cuadrado usando asteriscos.

Las líneas 2Ø y 3Ø imprimen las dos filas horizontales de asteriscos, y las líneas 4Ø y 5Ø se encargan de las dos columnas de asteriscos verticales.

Cuando se llena la pantalla: CLS y SCROLL

La pantalla del Spectrum puede representar treinta y dos caracteres por línea y veintidós líneas (es de 32 caracteres × 22 líneas), y antes o después acabará llenándola completamente, y

querrá imprimir más información todavía. Si ya ha terminado de ver lo que hay en la pantalla, puede usar la sentencia CLS (borrar pantalla) que simplemente borrará todo lo que había en la pantalla, y volverá a colocar la posición de impresión en el lado izquierdo de la fila superior de la misma.

Si está escribiendo cosas en la pantalla siguiendo la dirección de arriba a abajo, llegará un momento en que alcance la línea inferior. Si intenta imprimir una nueva línea, el Spectrum escribirá el mensaje: "Scroll? y/n", y esperará su respuesta. Si pulsa cualquier tecla que no sea "n", SPACE o STOP, la pantalla completa será desplazada hacia arriba una línea, la línea superior anterior se perderá y la nueva información se imprimirá en la línea inferior que se acaba de crear. Esta técnica de mover la pantalla entera una línea hacia arriba se conoce con el nombre de **«scrolling»,** y de ella hablábamos brevemente en el capítulo 2, cuando nos referíamos al listado de los programas.

Después de este primer desplazamiento hacia arriba, se introducen nuevas líneas en la parte inferior de la pantalla, es decir, la pantalla es desplazada cada vez que PRINT intenta ejecutarse para cambiar algo en la parte inferior de la pantalla. Después de veintidós desplazamientos, la línea que se creó con el primero de ellos estará ocupando ahora la parte superior de la pantalla, a punto de desaparecer, y el Spectrum preguntará de nuevo si usted está de acuerdo con un nuevo desplazamiento. De esa manera, el Spectrum se asegura de que tenga siempre la oportunidad de ver cada pantalla llena de números antes de que empiecen a desaparecer por la parte superior. Para ver cómo funciona, introduzca este programa:

```
1Ø PRINT RND
2Ø GOTO 1Ø
```

Si por cualquier motivo desea hacer que la pantalla se mueva hacia arriba antes de lo que normalmente lo haría, puede usar:

```
PRINT AT 21,Ø"
```

que mueve primero la posición de escritura a la línea 21, y después intenta moverla una línea más.

Control de INPUT

Puede resultar chocante, descubrir que la sentencia INPUT es capaz de utilizar todos los elementos que pueda usar la

sentencia PRINT. Este es un descubrimiento agradable, puesto que incrementa en gran manera la gama de comentarios de entrada que podemos escribir.

La forma general de la sentencia INPUT es:

INPUT "lista de impresión"

Existen, no obstante, algunas reglas adicionales e importantes diferencias en la forma de interpretarse la "lista de impresión", cuando se utiliza en una sentencia INPUT. El primer cambio consiste en que es necesario indicar qué variables son las que vamos a imprimir como parte de un comentario de entrada, y cuáles las que van a ser leídas con la sentencia INPUT. La regla que el Spectrum emplea es: cualquier "componente de la lista de impresión" que empiece por una letra, será tomada como una de las variables a las que vamos a asignar valor con INPUT (obsérvese que AT y TAB no empiezan con una letra, porque se obtienen con una única pulsación de tecla). Por ejemplo:

```
1Ø INPUT "a=";a;"b=";b
```

va a escribir un comentario de entrada; después espera a que usted introduzca un valor para a, imprime un segundo comentario y espera a que introduzca un valor de b.

Se puede evitar la regla de la primera letra, utilizando paréntesis para encerrar cualquier variable o expresión. El valor de una expresión no varía si encerramos ésta entre paréntesis (innecesarios), pero, en cambio, comenzará con una letra. Veámoslo:

```
1Ø LET a=1Ø
2Ø INPUT "a=";(a);"b=";b
3Ø PRINT a,b
```

Esto escribe el valor de a en la línea inferior y espera a que se introduzca un valor para b.

Ya habrá notado la extraña forma de trabajar de PRINT e INPUT en partes separadas de la pantalla. Esta división de la pantalla afecta al modo de funcionamiento de la función AT. Si se usa AT en una sentencia PRINT, las líneas se cuentan empezando en la parte superior de la pantalla, siendo la primera línea la número Ø. Sin embargo, si se emplea AT en una sentencia

INPUT, se deben empezar a contar las líneas desde la parte inferior de la pantalla, empezando en cero, que es la primera línea del área de entrada (ver figura 7.2). Por ejemplo, pruebe con:

```
1Ø INPUT AT Ø,5;"linea superior";AT 1,5;"linea siguiente
";a
```

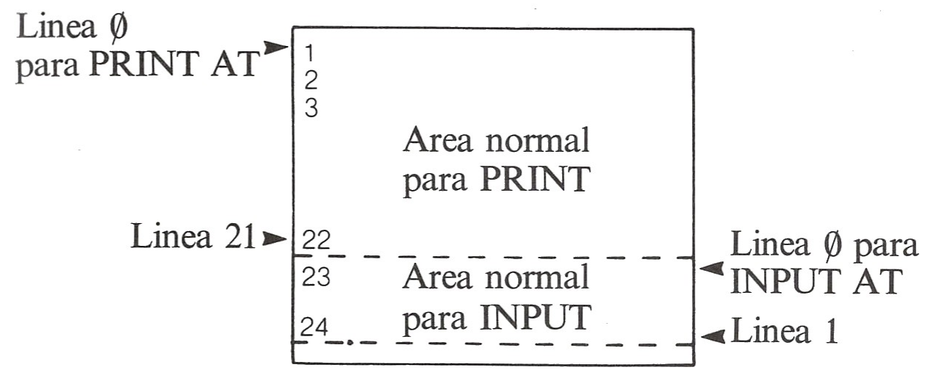


Fig. 7.2. Areas de la pantalla.

La última sentencia dada escribe dos comentarios, uno en cada línea del área de entrada, y después espera a que se introduzca un valor para "a". Tal vez a estas alturas se pregunte qué ocurre si intenta escribir en líneas del área de entrada diferentes de Ø y 1. La respuesta es que el Spectrum desplaza el área de entrada hacia arriba para hacer sitio a la nueva línea de comentarios de INPUT. Se puede observar con:

```
1Ø INPUT AT 1Ø,Ø;"esta linea esta normalmente fuera de la
pantalla";
AT Ø,Ø;"y esta es la línea superior";a
```

Si hay cualquier cosa escrita en la parte superior de la pantalla, y un INPUT quiere imprimir algo en el mismo lugar, el área de impresión (de PRINT) de la pantalla se desplaza hacia arriba, para dejar espacio al área de entrada. Esto es más sencillo de comprender después de haber visto cómo ocurre:

```
1Ø PRINT AT 15,Ø;"Esto esta escrito en la linea 15"
2Ø PRINT AT 16,Ø;"Esto esta escrito en la linea 16"
3Ø INPUT AT 2Ø,Ø;"Esto esta escrito en la línea 2Ø del area
de entrada";AT Ø,Ø;"esta es la linea superior del area de entra-
da";a
```

Se puede tener una sentencia INPUT que sólo imprima en la pantalla, pero no es probable que pueda ver el mensaje escrito

por ella, puesto que **toda** sentencia INPUT borra el área de entrada cuando termina de ejecutarse. Eso significa que su mensaje parpadeará un instante en la pantalla, que es borrada a continuación, y el Spectrum pasará a ejecutar la siguiente sentencia. Por otro lado, no sólo toda sentencia INPUT borra el área de entrada, sino que además restablece el tamaño original de ésta, es decir, las dos líneas de la parte baja de la pantalla.

La división de la pantalla en dos áreas distintas, la de impresión y la de entrada, algunas veces se ha criticado como si fuese un problema del Spectrum (y del ZX-81); pero si usted entiende cómo funciona, podrá convertir el problema en una ventaja.

Los caracteres gráficos

La combinación de PRINT y AT puede utilizarse, evidentemente, para colocar un carácter en cualquier lugar de la pantalla, y, como vimos antes en este capítulo (en el programa que pintaba un cuadrado), eso se puede aprovechar para producir gráficos limitados. Esta capacidad se pone de manifiesto realmente, cuando se combina con el conjunto de caracteres gráficos del Spectrum. Estos caracteres pueden obtenerse presionando CAPS SHIFT y GRAPHICS (9), para cambiar el cursor a □, y pulsando después otra tecla. Puede dejar el modo gráfico presionando de nuevo GRAPHICS. Aunque siempre que se pulse una tecla aparece algo en la pantalla, en modo gráfico hay únicamente treinta y siete caracteres gráficos. Ocho de ellos están impresos en color blanco en la fila superior del teclado, junto con los números. Se pueden obtener ocho más, pulsando cualquiera de las teclas de la fila superior junto con CAPS SHIFT. Estos últimos caracteres son los ocho primeros, pero en modo invertido: blanco cambiado por negro y viceversa; pero es importante recalcar que son caracteres diferentes. Este conjunto de dieciséis caracteres gráficos también puede imprimirse en la pantalla, usando sus códigos de carácter en la función CHR$. Para ver el juego completo de los dieciséis caracteres aludidos, introduzca el programa siguiente:

```
10 FOR i=128 TO 136
20 PRINT i,CHR$(i)
30 NEXT i
```

Los otros veintiún caracteres gráficos se obtienen al pulsar las teclas de la "A" a la "U", cuando estamos en modo gráfico y

corresponden a los códigos de carácter de 144 a 164. Como después explicaremos, estos caracteres pueden cambiarse para introducir cualquier forma que se desee, esto es, son gráficos definidos por el usuario, aunque, inicialmente, producirán una copia del conjunto de caracteres de letras mayúsculas. Por ello, si usted pulsa "A" en modo gráfico, aparecerá una "A" mayúscula en la pantalla.

Aparte de estos treinta y siete caracteres mencionados, si se pulsa una tecla en modo gráfico obtendremos simplemente un carácter de los que se pueden producir en cualquiera de los otros modos del cursor. Para especificar más: cuando estamos en modo gráfico, al pulsar una tecla cualquiera introducirá el carácter que se encuentre noventa y seis lugares más abajo en la lista de caracteres del que se obtiene con dicha tecla normalmente. En otras palabras, si la tecla produce el carácter CHR$(n) normalmente, ahora, en modo gráfico, producirá CHR$(n+96).

Los caracteres gráficos se comportan como cualquier otro carácter, si dejamos a un lado que hay que introducirlos en modo gráfico. Por ejemplo, se pueden emplear en las sentencias PRINT como cadenas corrientes. El programa que dibujaba un cuadrado con asteriscos se puede modificar para que su resultado se parezca más a un cuadrado real usando los caracteres gráficos:

```
10 FOR i=1 TO 9
20 PRINT AT 5,i+10;"[3]"
30 PRINT AT 15,i+10;"[3]"
40 PRINT AT i+5,10;"[5]"
50 PRINT AT i+5,20;"[5]"
60 NEXT i
```

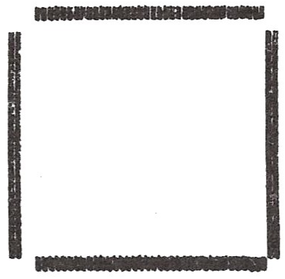


Fig. 7.3. Cuadrado usando caracteres gráficos.

(Como ya explicamos en el capítulo 2, debido a la dificultad de imprimir caracteres gráficos en un libro, se representarán por el carácter principal de la tecla encerrado entre paréntesis cua-

drados. Por ejemplo: [1] es el carácter gráfico de la tecla "1". Si, además, el carácter debe introducirse pulsando a la vez CAPS SHIFT y la tecla adecuada, escribiremos una "↑" antes de dicha tecla. Entonces, [↑1] es el carácter gráfico producido al pulsar la tecla "1", mientras se está pulsando CAPS SHIFT).

El problema del nuevo cuadrado que hemos dibujado, es que no tiene las cuatro esquinas, y hacer que aparezcan es cuestión de imprimir en el sitio adecuado los símbolos gráficos que tienen forma de "L". Dejamos esta tarea para que usted mismo la lleve a cabo.

Como puede suponer, dibujar formas más complejas con los carácteres gráficos es muy difícil. Afortunadamente, además de dibujar ocasionalmente líneas verticales u horizontales de gran anchura, los caracteres gráficos de la fila superior del teclado se utilizan normalmente agrupados en cadenas de pequeña longitud, para imprimir formas especiales. Por ejemplo, si desea imprimir la silueta de un barco durante el desarrollo de un juego, podría aprovechar la sentencia:

```
PRINT AT y,x; "[↑2][↑3][↑3]"
```

y aparecería un barco en la columna x de la línea y:

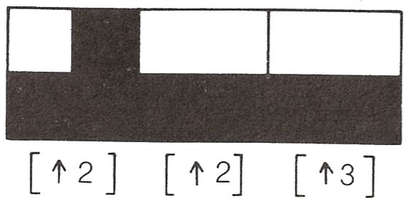


Fig. 7.4. Caracteres gráficos de barco.

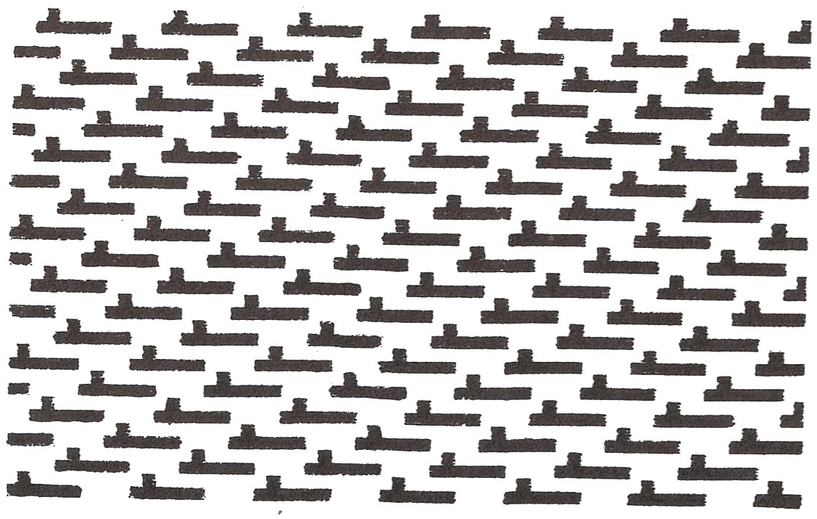


Fig. 7.5. Gráficos de barcos.

Para poder ver un barco moviéndose por la pantalla, introduzca

```
10 FOR x=0 TO 27
20 PRINT AT 5,x;" [↑2][↑3][↑3]"
30 NEXT x
```

Observe el espacio en blanco dejado antes de escribir el primer carácter gráfico de la línea 20. Si desea saber para qué vale: ¡pruebe el programa quitándolo!

Caracteres gráficos definidos por el usuario

La gama de formas que pueden obtenerse con combinaciones de caracteres gráficos de la fila de teclas superior del Spectrum, que discutimos en la sección anterior, es bastante limitada. Por ejemplo: ¿cómo podríamos dibujar con ellos la forma de un hombre? Afortunadamente, el Spectrum posee veintiún caracteres gráficos, que pueden alterarse para obtener cualquier forma que desee.

Antes de meternos en profundidad con la forma en que se definen estos nuevos caracteres, tenemos que examinar primero cómo se producen los caracteres en la pantalla. Cada uno de los caracteres que el Spectrum puede imprimir en la pantalla está compuesto por una cuadrícula de sesenta y cuatro puntos, dispuestos en un cuadrado de 8×8 puntos. La forma de cualquier carácter depende de qué puntos de la cuadrícula sean de color negro en la pantalla y cuáles sean blancos. Por analogía con la impresión o la escritura en un papel, los puntos negros se denominan "tinta" (INK en inglés), y los puntos blancos "papel" (PAPER, en inglés). Por ejemplo, la letra "a" se produce con la disposición de puntos negros y blancos que puede verse en la figura 7.6, donde "t" corresponde a tinta (ink) y "p" a papel (paper).

```
p p p p p p p p
p p p p p p p p
p p i i i p p p
p p p p p i p p
p p i i i i p p
p i p p p i p p
p p i i i i p p
p p p p p p p p
```

Fig. 7.6.

Podría resultar algo difícil ver la forma de la letra "a" entre las letras "t" y "p" de la cuadrícula; pero la "a" se percibirá claramente, si cambiamos la "t" por un asterisco y la "p" por un blanco, como muestra la figura 7.7.

```
 * * *
       *
 * * * *
*       *
 * * * *
```

Fig. 7.7.

Obviamente, si vamos a definir la forma que corresponderá a un "carácter gráfico definido por el usuario", debe de existir alguna manera de especificar qué puntos de la cuadrícula de 8×8 van a ser la tinta (negros) y cuáles serán el papel (blancos). El Spectrum nos proporciona un camino muy simple para lograr eso, por medio de las funciones USR y BIN. El carácter definido por el usuario que va a reemplazar la definición actual del carácter [n], donde n es una letra mayúscula de las que hay entre "A" y "U", tiene que ver con la función:

USR "n"

Por ejemplo, si vamos a definir un nuevo carácter con la forma de un hombre, y queremos que se obtenga pulsando [M] (M en modo gráfico), se utilizará la función:

USR "M"

La definición del nuevo carácter debe hacerse fila por fila. Cada fila de la cuadrícula (fila es una línea horizontal), puede escribirse como una secuencia de ocho dígitos poniendo un Ø para los puntos que serán papel, y un 1 para los que serán la tinta. Vamos a ver un caso: la fila pptttppp se escribirá como la secuencia ØØ111ØØØ. Mediante esta representación, cada fila del nuevo carácter puede ser redefinida con:

POKE USR "M"+r,BIN "configuración de puntos"

donde "r" es el número de la fila que vamos a definir (la primera fila tiene el número Ø) y "configuración de puntos" es la secuencia de ceros y unos que corresponden a la distribución de la tinta y de papel, para los puntos de esa fila. El comando POKE y la función BIN se verán en el capítulo 10; aunque lo que hacen exactamente no tiene importancia para definir los caracteres (para este fin siempre se utilizan de igual manera). Para definir un carácter completo, tiene que especificar las ocho filas y, por tanto, emplear la sentencia anterior ocho veces. Pero recuerde: no hay nada que le obligue a cambiar a la vez las ocho filas de un carácter (podría aprovechar alguna fila de las originales).

Todo esto será más fácil de comprender después de un ejemplo. El carácter con forma de hombre que comentábamos al empezar la sección, podría definirse con la siguiente configuración de puntos:

```
ØØØ11ØØØ
ØØØ11ØØØ
11111111
ØØ1111ØØ
ØØ1111ØØ
ØØ1ØØ1ØØ
ØØ1ØØ1ØØ
ØØ1ØØ1ØØ
```

donde se ha usado un 1 para indicar "tinta", y un Ø para indicar "papel". Esto puede transferirse a la tecla M (en modo gráfico), [M], fila por fila:

```
1Ø POKE USR "M"    ,BIN ØØØ11ØØØ
2Ø POKE USR "M"+1,BIN ØØØ11ØØØ
3Ø POKE USR "M"+2,BIN 11111111
4Ø POKE USR "M"+3,BIN ØØ1111ØØ
5Ø POKE USR "M"+4,BIN ØØ1111ØØ
6Ø POKE USR "M"+5,BIN ØØ1ØØ1ØØ
7Ø POKE USR "M"+6,BIN ØØ1ØØ1ØØ
8Ø POKE USR "M"+7,BIN ØØ1ØØ1ØØ
9Ø PRINT "[M]"
```

Observe cómo una vez que ha ejecutado este programa, la [M] de la línea 9Ø se imprime con la forma de un "hombrecito". La razón de esto es que, una vez que usted define un carácter gráfico, la definición permanece hasta que vuelva a cambiar el carácter o apague la máquina.

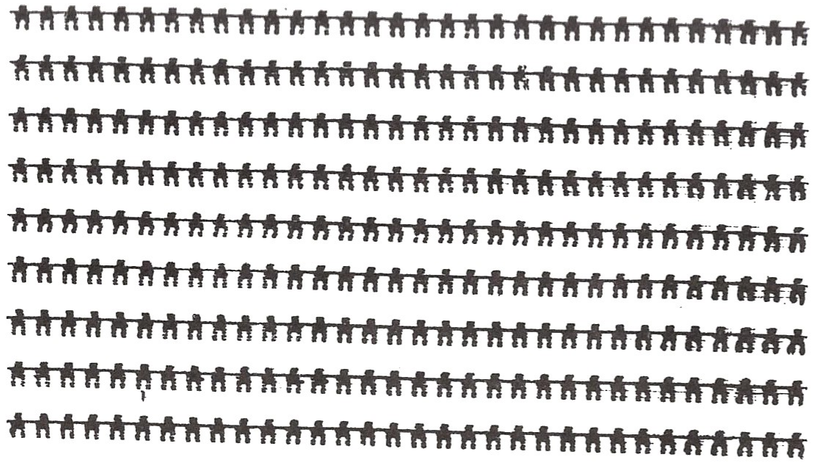


Fig. 7.8. Gráficos de hombre.

Los caracteres gráficos definidos por el usuario, son los más útiles para producir las figuras especiales que son tan esenciales para cualquier tipo de programa de juegos. Por ejemplo, un carácter en forma de punto grande podría usarse como una pelota. Sin embargo, hay algunos usos serios de los gráficos de usuario tal como imprimir en pantalla fórmulas matemáticas o químicas.

Cambio de aspecto de los caracteres: INVERSE y OVER

Los comandos INVERSE y OVER no crean, ni nos dan acceso a ningún carácter nuevo, sino que pueden usarse para alterar la forma en que los caracteres existentes aparecen en la pantalla. El comando INVERSE es, quizás, el más fácil de entender. El efecto de:

INVERSE 1

es intercambiar los puntos de "tinta" y "papel" de cualquier carácter que se imprima a continuación, es decir, los puntos de tinta originales pasan ahora a ser papel, y viceversa. Esto quiere decir que en vez de los caracteres negros sobre fondo blanco normales, obtendrá ahora caracteres blancos con fondo negro. Para regresar a la asignación de tinta y papel de siempre, debe usarse el comando:

INVERSE Ø

Puede pensar en el Ø y el 1 que siguen a INVERSE como "desactivado" y "activado", respectivamente. Por ejemplo:

```
1Ø INVERSE 1
2Ø PRINT "esto esta invertido"
3Ø INVERSE Ø
4Ø PRINT "esto esta otra vez normal"
```

Aunque INVERSE es un comando fácil de entender, hay dos puntualizaciones que deben hacerse: INVERSE no crea ningún carácter nuevo, sino que más bien cambia la definición de los que ya existían, puesto que hace que los puntos de tinta pasen a ser ahora de papel y viceversa. El efecto de INVERSE 1 permanece hasta que el Spectrum ejecute un INVERSE Ø, o hasta que se desconecte. El resultado de esta permanencia del comando INVERSE es que antes de desactivar la inversión de caracteres en un programa, todos los listados, etc., aparecerán en la pantalla en modo invertido. La solución para este problema es introducir un comando INVERSE Ø en modo inmediato, o sea, sin número de línea.

El comando OVER es muy similar a INVERSE, puesto que también altera la forma en que los puntos de tinta y de papel aparecen en la pantalla. Pero es ligeramente más difícil de explicar. Después del comando:

OVER 1

el que un punto de tinta aparezca en la pantalla o no, dependerá de lo que hay ya en ésta. Si el nuevo punto y el anterior eran ambos de tinta o ambos de papel, el punto correspondiente aparecerá como papel. Si el nuevo punto y el antiguo eran diferentes, el punto en cuestión aparecerá ahora como de tinta. Puede parecer complejo, pero es fácil de resumir:

Nuevo punto	*Anterior punto*	*Punto en la pantalla*
Papel	Papel	Papel
Papel	Tinta	Tinta
Tinta	Papel	Tinta
Tinta	Tinta	Papel

y puede recordarse con la regla: "dos cosas iguales dan papel, y una de cada dan tinta". (Este comportamiento se llama O-

exclusivo del punto anterior con el nuevo. Vea el capítulo 10.) Para ver el efecto que produce, pruebe esto:

```
1Ø OVER 1
2Ø PRINT AT Ø,Ø;"Que bonito"
3Ø PRINT AT Ø,Ø;"- - - - - - - - - -"
```

Ahora, usted podrá ver el mensaje producido por la línea 2Ø subrayado. La forma en que funciona esto es: la fila inferior de puntos de cada carácter es una fila de puntos de papel, y el carácter "subrayado" es simplemente una fila inferior de puntos de tinta. Por las reglas que hemos dato antes, papel y tinta dan tinta, y por eso aparece el subrayado en la fila inferior de los caracteres que ya estaban impresos en la pantalla.

El motivo de llamar OVER a este comando es que su efecto es muy similar a la sobreimpresión (over-printing, en inglés) en una máquina de escribir. Sin embargo, es importante resaltar que en otras muchas cosas OVER no se parece en nada a la sobreimpresión en máquinas de escribir. Por ejemplo, introduzca:

```
1Ø OVER 1
2Ø PRINT AT Ø,Ø;"O"
3Ø PRINT AT Ø,Ø;"/"
```

En una máquina de escribir, esto hubiese producido una letra "O" con una barra "/" (slash) atravesándola; pero como dos puntos de tinta dan un punto de papel, los puntos donde la "O" y la barra "/" son ambos negros, aparecen como puntos blancos de papel.

Aunque esto parezca un inconveniente, puede utilizarse para crear curiosos efectos en pantalla. Vea este nuevo programa:

```
1Ø OVER 1
2Ø PRINT AT Ø,Ø;"*"
3Ø GOTO 2Ø
```

La primera vez que se imprime, el asterisco aparece en la pantalla porque todos los puntos anteriores eran papel. La segunda vez desaparece, porque todos los nuevos puntos de tinta van al mismo lugar que los puntos de tinta anteriores (imprimimos un asterisco sobre otro) y, por tanto, se cancelan. Esto se

repite cada vez que el asterisco vuelve a imprimirse y, en consecuencia, obtenemos el efecto de parpadeo que se observa.

Es importante comentar que OVER es lo mismo que INVERSE, en cuanto a que tampoco crea ningún carácter nuevo, sino que altera la forma en que los caracteres existentes van a aparecer en la pantalla, y sus efectos también permanecen hasta que sean anulados con:

```
OVER Ø
```

o se desconecte la máquina.

Atributos de los caracteres: FLASH y BRIGHT

Hasta el momento nos hemos fijado en la posición y la forma de los caracteres impresos en la pantalla. Sin embargo, hay otras cosas que se pueden controlar en cuanto a la forma en que el Spectrum presenta los caracteres en la televisión. Los comandos FLASH y BRIGHT pueden utilizarse para alterar la forma en que cualquier carácter aparece en pantalla. El comando

```
FLASH 1
```

hace que los siguientes caracteres que se escriban **parpadeen** (aparezcan "intermitentes"); y:

```
BRIGHT 1
```

provocará que los caracteres que se impriman a continuación, tengan **mayor brillo** que los normales. Como para INVERSE y OVER, se pueden anular los efectos de FLASH y BRIGHT usando un Ø, en lugar del 1, en los comandos. Estos dos comandos son mucho más fáciles de emplear que de explicar. Pruébelos con el siguiente programa:

```
1Ø FLASH 1
2Ø PRINT "caracteres parpadeantes"
3Ø FLASH Ø
4Ø BRIGHT 1
5Ø PRINT "caracteres brillantes"
6Ø BRIGHT Ø
```

El texto escrito por la línea 2Ø parpadeará (el negro cambia a blanco y el blanco a negro sucesivamente) y el mensaje producido por la línea 5Ø aparecerá más brillante que el resto de la pantalla. Como ya habría supuesto, el efecto de FLASH y BRIGHT permanece hasta que se desactiven o se apague el computador.

Color: BORDER, INK y PAPER

A lo mejor le sorprende que hayamos tardado tanto tiempo en llegar al excitante tema del color. La causa de la tardanza es que la forma en que trabajan los comandos de color del Spectrum, es mucho más sencilla de entender una vez que se conoce ya el concepto de puntos de tinta y de papel de un carácter.

El Spectrum puede manejar hasta ocho colores diferentes en la pantalla del televisor, y los colores se designan con números del Ø al 7:

Ø - negro
1 - azul
2 - rojo
3 - magenta (púrpura)
4 - verde
5 - azul pálido
6 - amarillo
7 - blanco

Sin embargo, no necesitará aprenderse de memoria estos números, porque los nombres de los colores están escritos (con el propio color que indica cada nombre) encima de las teclas de los números. Observe cómo los números más pequeños corresponden a los colores más oscuros. Si usted no está usando un televisor en color con su Spectrum, los colores aparecerán como diferentes tonalidades de gris, correspondiendo los tonos más oscuros, como era de esperar, a los números menores.

Tal vez la forma más sencilla de ver la gama de colores del Spectrum sea con el comando BORDER. El área de la pantalla de televisión en que el Spectrum no puede imprimir se denomina **borde** (border), y, aunque no se puede escribir en él con PRINT, se puede determinar su color (el borde consta de la parte superior, los extremos laterales y la parte inferior de la pantalla, considerada esta última hasta la última línea del área de entrada). El comando:

```
BORDER c
```

cambiará el color del borde por el color que corresponde al número "c" (en general, "c" puede ser una expresión aritmética). Pruebe el programa siguiente:

```
1Ø FOR i=Ø TO 7
2Ø INPUT a$
3Ø BORDER i
4Ø NEXT i
```

que cambiará el color del borde de la pantalla cada vez que pulse ENTER. Para comprobar que la zona de entrada está incluida en el borde, cambie la línea 2Ø por ésta:

```
2Ø INPUT AT 2Ø,Ø;a$
```

Del mismo modo que se puede cambiar el color del borde, se puede cambiar el color del resto de la pantalla. No obstante, a diferencia del borde, esto lleva implícito el especificar **dos colores:** uno para los puntos de tinta y otro para los puntos de papel. Cuando se conecta el Spectrum, los puntos de tinta son de color negro y los puntos de papel son de color blanco, pero se puede alterar esta asignación de colores usando:

```
INK c
```

y

```
PAPER c
```

El comando INK establece el color en que aparecerán los puntos de tinta, y PAPER establece el color de los puntos de papel que se impriman. Por ejemplo, ejecute el programa:

```
1Ø INPUT "color de la tinta= ";t
2Ø INPUT "color del papel= ";p
3Ø INK t
4Ø PAPER p
5Ø PRINT "*";
6Ø GOTO 1Ø
```

Con este programa se pueden imprimir asteriscos de cualquier color de tinta y de papel (introduciendo los números de colores que piden las líneas 1Ø y 2Ø). Hay que destacar que el Spectrum siempre permitirá que usted especifique el mismo co-

lor para la tinta y el papel, en cuyo caso, ¡no verá el asterisco! Pero diremos que, aunque no pueda ver el asterisco, éste está aún ahí, en la pantalla (la forma de los puntos del carácter está presente en la pantalla, pero no puede verse, porque los puntos de tinta y de papel son del mismo color).

Estos son, casi, los únicos comandos de color de que dispone el Spectrum, pero, utilizándolos, es posible crear algunos efectos de color excelentes. Sin embargo, es importante conocer las limitaciones que tiene este método sencillo de controlar el color. Cada bloque de ocho por ocho que constituye un carácter puede aparecer **únicamente** en dos colores distintos: el color de papel y el de tinta. Eso significa que, aunque en la pantalla podemos tener los ocho colores al mismo tiempo, sólo podrán existir dos colores distintos dentro de una única posición de carácter. Esta característica se convierte en un problema cuando se usan gráficos de alta resolución, como veremos en el capítulo 9.

Además de los colores de Ø a 7, existen también dos **pseudocolores,** que corresponden a los números 8 y 9. Si se emplea el número 8 en cualquiera de los comandos INK, PAPER, BRIGHT o FLASH, el efecto será que el estado en que se encuentre cualquier posición de carácter, permanece invariable durante las siguientes impresiones. Por ejemplo, si se ejecuta PAPER 8, cualquier futura impresión usará el color del papel que ya existía en el lugar donde va a escribir. FLASH 8 dejará **«parpadeantes»** las posiciones de carácter que estaban ya en ese estado, y mantendrá sin parpadeo a las que estaban sin él. Debido a que, en cierto modo, los comandos que usan el color 8 permiten que los colores originales (o las condiciones de presentación en pantalla) se sigan manteniendo, a veces, se denomina a este color: **color transparente.**

El color 9 sólo puede emplearse con los comandos INK y PAPER, y simplemente indica al Spectrum que use un color que **contraste** con el color que estaba presente en la posición de impresión. En la práctica, se usa negro para contrastar con los colores claros (de 4 a 7), y blanco para contrastar con colores oscuros (de Ø a 3). Para verlo pruebe con:

```
1Ø INK 9
2Ø FOR c=Ø TO 7
3Ø PAPER c
4Ø PRINT c
5Ø NEXT c
```


Los comandos de impresión y el color

Los demás comandos de impresión, OVER, INVERSE, FLASH y BRIGHT, funcionan de una manera que permanece inalterada, sean cuales sean los colores establecidos para el papel y la tinta. Por ejemplo, INVERSE intercambiará los puntos de papel y tinta, sin preocuparse de los colores que éstos tengan. Una diferencia es que BRIGHT en blanco y negro parece afectar únicamente a los puntos de papel. Esto es debido a que los puntos de papel son blancos y, por tanto, pueden ser mostrados con mayor brillo de lo normal, pero los puntos de tinta son negros y la idea de un negro brillante, ¡es un poco rara! Sin embargo, en lo que respecta a los demás colores, BRIGHT se comporta como siempre. Puede emplear también cualquier combinación de los comandos, y los resultados son, por lo general, fáciles de predecir. Por ejemplo, introduzca:

```
1Ø BRIGHT 1
2Ø FLASH 1
3Ø PRINT "parpadeo brillante"
```

Si piensa en términos de puntos de tinta y de papel, que dan forma a los caracteres en la pantalla, y los comandos INK y PAPER, que establecen los colores en que aparecen los puntos de tinta y de papel, entonces no tendrá dificultades para comprender el funcionamiento de los colores del Spectrum. Como un ejemplo de lo lógico que resulta todo, considere la sentencia CLS que vimos anteriormente. CLS borra el contenido de la pantalla rellenándola con puntos de papel, por lo que, si quiere poner toda la pantalla de un color, el siguiente programa es adecuado para ello:

```
1Ø INPUT "color=";c
2Ø PAPER c
3Ø CLS
4Ø GOTO 1Ø
```

Cuando lo haga, verá cómo la pantalla cambia de color instantáneamente.

Colores temporales

Obviamente, sería una ventaja poder cambiar únicamente los colores de los caracteres escritos por una sentencia PRINT. Esto

se puede conseguir incluyendo INK o PAPER como parte de la lista de PRINT. Por ejemplo:

```
1Ø PAPER 3
2Ø INK 6
3Ø CLS
4Ø PRINT "COLOR 6"
5Ø PRINT INK 4;"COLOR 4"
6Ø PRINT "COLOR 6"
```

Cualquiera de los comandos de control de impresión que ya conocemos puede usarse de esta manera, es decir, cualquiera de la lista: INK, PAPER, FLASH, BRIGHT, OVER o INVERSE, puede emplearse como parte de una "lista de impresión", y sus efectos se limitan a los caracteres que imprime la sentencia PRINT en la que están. Las mismas consideraciones son ciertas para la "lista de impresión" de las sentencias INPUT.

El uso de tales comandos en una sentencia PRINT es, normalmente, la mejor forma de controlar las salidas en la pantalla para todos los programas, salvo las más simples. Establezca los colores globales de tinta y de papel y después, cada vez que desee imprimir algo en otro color, inserte los comandos de color en una sentencia PRINT. De esta manera, siempre sabrá qué colores aparecerán en la pantalla.

Uso de los gráficos en los juegos

Ahora estamos en posición de usar gráficos para aplicaciones de programación. No obstante, la utilización en los programas de lo que hemos aprendido en este capítulo, se dejará para más adelante, cuando hayamos explicado el sonido, un elemento indispensable para escribir juegos excitantes.

Capítulo 8
Sonido y juegos

El comando para producir sonido en el Spectrum es muy simple. Sin embargo, se necesita mucho ingenio para producir cualquier sonido digno de escucharse. En la primera parte de este capítulo examinaremos algunas de las ideas implicadas en el uso del sonido para lograr efectos agradables. En la segunda parte, presentaremos y explicaremos un ejemplo de juego que incorpora sonido y gráficos. Aunque la mayor parte de este juego se podría haber escrito al final del capítulo 7, es notable la cantidad de mejoras que se pueden añadir a un juego con un manejo cuidadoso del sonido.

Sonidos sencillos: BEEP

El único comando de sonido del Spectrum es:

BEEP "expresión artimética 1", "expresión aritmética 2",

donde el valor de "expresión aritmética 1" especifica el tiempo en segundos, y "expresión aritmética 2" especifica el tono en semitonos, por encima o por debajo del DO natural. Por ejemplo:

```
BEEP 1,Ø
```

"tocará" un DO natural durante un segundo exactamente. Fíjese en que todos los cálculos se detendrán mientras está sonando la nota.

Para escuchar la gama de notas que tiene a su disposición, pruebe:

```
10 FOR i=69 TO -60 STEP -1
20 BEEP .1,i
30 NEXT i
```

La calidad de las notas muy altas es muy pobre, son más un "pitido" que una nota. Las notas más bajas al principio suenan como un ruido áspero y después como una serie de "clicks". Esto no es nada raro, pues "el click" es realmente el único sonido que puede producir el Spectrum. Los tonos regulares son una sucesión muy rápida de estos "clicks" (veremos después, en este capítulo, cómo obtener esos "clicks" elementales desde BASIC).

Si usted sabe un poco de música, puede usar BEEP para escribir sus propias canciones. Sin embargo, si antes que nada le gustaría escuchar lo que el Spectrum puede hacer por sí solo, ¡ahí va eso!:

```
10 BEEP .1,50-INT(RND*100)
20 GOTO 10
```

El ruido que produce este programa es interesante al principio, pero pronto se vuelve aburrido. El problema es que la música demasiado aleatoria no suena bien. Aunque es muy difícil introducir en la música generada por un computador suficiente **orden,** como para que suene parecida a la música tradicional, puede ver el efecto global de imponer un cierto orden, si introduce el siguiente programa:

```
10 LET n=0
20 LET n=n+SGN(1-2*RND)
30 BEEP .01,n
40 GOTO 20
```

Esto interpreta una larga sucesión de notas que suben o bajan un semitono como máximo y "suenan" a música algo más que el primer programa. De hecho, no es una mala imitación del "Vuelo del moscardón".

Antes de abandonar el tema de la música aleatoria, es interesante escuchar el efecto de cambiar la duración de la nota, en cada uno de los programas anteriores. Pruebe a sustituir, en la línea 30, el 0.01 por 1,0.5, 0.1 segundos. En vez de música aleatoria, podría ocurrírsele la idea de que "interpretar" en el

Spectrum es más interesante, y realmente no es difícil convertir el teclado del Spectrum en un teclado musical. Escriba el programa:

```
10 DATA 0,2,4,5,7,9,11,12
20 DIM n(8)
30 FOR i=1 TO 8
40 READ n(i)
50 NEXT i
60 LET a$=INKEY$
70 IF a$="" THEN GOTO 60
80 BEEP .3,n(VAL(a$))
90 GOTO 60
```

Esto le permitirá interpretar notas con las teclas numéricas de 1 a 8 en la fila superior del teclado. Funciona **explorando continuamente** el teclado con la función INKEY$ (ver capítulo 6). Cada número se introduce como una variable de cadena, y después se convierte de cadena a número, por medio de la función VAL; posteriormente se emplea dicho número en la sentencia BEEP (línea 80) para controlar el tono. La tabla "n" guarda los valores de tono para cada nota: desde DO mayor hasta un DO una octava superior.

Se podría escribir un programa que hiciese que cualquiera de las teclas produjese una nota diferente. Lo que ya es más difícil es encontrar una distribución de las notas en las teclas, que haga del Spectrum un "instrumento" fácil de tocar. Si quiere escuchar la salida del Spectrum un poco más alta, debe saber que la señal de sonido está, también, presente en los conectores para el magnetofón. Si, simplemente, coloca en posición de grabación, el magnetofón que utiliza para cargar y grabar programas, cuando el Spectrum está "interpretando" algo, podrá rebobinar la cinta y escuchar la "melodía" con mayor volumen, a través del magnetofón. En el caso de que sea un entusiasta del sonido del Spectrum, nada le impide conectar un amplificador externo en la toma MIC del computador.

Programación de melodías

Para programar canciones, ya sean conocidas o compuestas por usted mismo, tiene que averiguar la secuencia de notas y la

duración de cada una. Esto resulta fácil si tiene la canción escrita. Si sabe encontrar DO natural en el pentagrama musical, una nota escrita en esa línea corresponde a un tono de Ø.

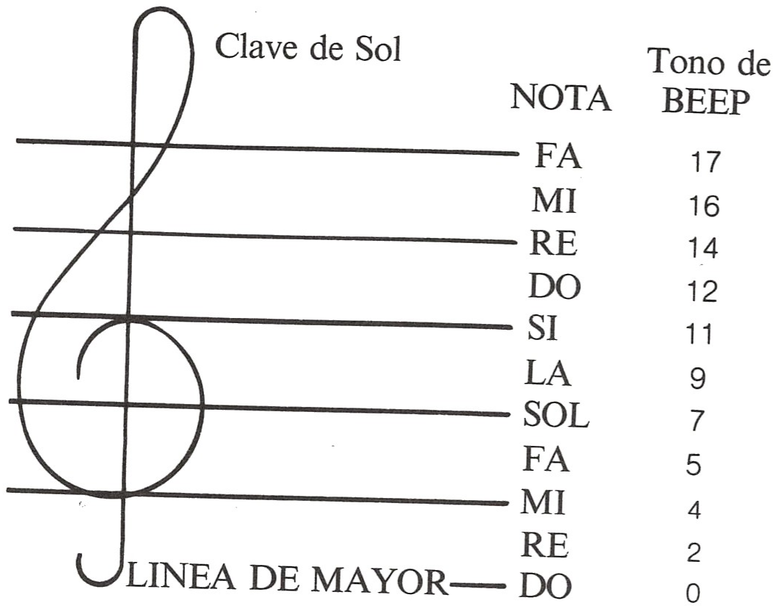


Fig. 8.1.

Si nos movemos un lugar hacia arriba o hacia abajo del pentagrama, el tono se incrementa o decrementa en 2 ó 1 respectivamente (ver figura 8.1). Esto se debe a que las notas difieren unas veces en un tono completo y otras veces sólo en un semitono. La secuencia de diferencias tono/semitono entre las notas es fácil de recordar, porque es exactamente la misma que la disposición de teclas negras y blancas en el piano.

Por ejemplo: empezando en DO, tenemos la siguiente secuencia de diferencias tono/semitono:

DO - RE - MI - FA - SOL - LA - SI - DO
T T S T T T S

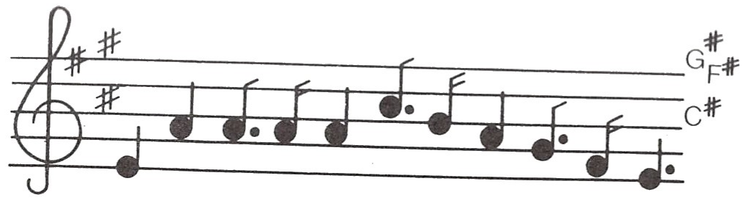
Un problema adicional es que la música comprende también bemoles y sostenidos. Ambas son fáciles de manejar, si piensa que un sostenido eleva en 1 el valor de la nota, y un bemol disminuye en 1 el valor de dicha nota. Por ejemplo, DO es Ø; DO sostenido es 1 y DO bemol es −1. Lo único que debe recordar es que si una nota aparece como sostenido o bemol al comienzo de la música (es decir, en la clave que aparece al

comenzar una partitura), entonces esa nota y todas sus octavas deben ser sostenidas o bemoles.

Por ejemplo, el principio de "Hearts of Oak" (corazones de roble), además de ser una buena melodía, podría servir de base a un "tintineo" apropiado para un juego que tuviese que ver con los barcos. Las primeras once notas pueden verse en la figura 8.2, y transformarlas en valores de tono para BEEP es bastante fácil. Los tres signos de sostenido (#) del principio, se aplican a todos los SOL, FA y DO de la melodía; y esta regla se aplica mejor escribiendo el nombre de la nota debajo, y escribiendo un sostenido en cada SOL, FA y DO. Los valores de tono son asignados empleando la figura 8.1, y recordando añadir 1 a las notas sostenidas. Cuando transforme las canciones que tienen bemoles en su clave de tono, tiene que restar uno cada vez que se interprete la nota con bemol. Los valores de tono resultantes se pueden observar bajo el nombre de cada nota en la figura 8.2. Nos falta únicamente una cosa para poder escuchar la melodía propuesta, y es el problema de averiguar el tiempo durante el que debe interpretarse cada nota. Afortunadamente, la notación musical es rigurosamente lógica (después de todo, ¡es uno de los primeros lenguajes de programación!). El tiempo está dividido en intervalos, y una nota normal, como la primera de la partitura de la figura 8.1, debe durar un intervalo. El tiempo que dura una nota se acorta según el número de "barras" que aparecen en la "cola" que lleva dicha nota. Cada barra divide la longitud de la nota. Entonces, por ejemplo, la cuarta nota tiene dos barras: la primera la reduce a la mitad de un intervalo, y la segunda la reducirá a un cuarto de intervalo. La única complicación es que un punto que va a continuación de una nota es una instrucción para prolongarla la mitad del tiempo de lo que duraría normalmente (eso hace que uno se pregunte cómo se las arreglan los músicos). Por todo lo anterior, la tercera nota, que duraría la mitad de un intervalo, por estar seguida de un punto deberá tener una duración de un medio más un cuarto, es decir: tres cuartos. La transcripción de la notación musical a fracciones de intervalo de tiempo, da los resultados que aparecen escritos debajo de los valores de tono en la figura 8.2.

Ahí hay dos notas que no aparecen en "Hearts of Oak" y tienen que ser interpretadas el doble y el cuádruple de su duración respectivamente. Ambas se incluyen en la figura 8.3, junto con los valores de las demás notas.

¡Ha llegado la hora de empezar a programar! Cada nota de



MI	LA	LA	LA	LA	DO*	SI	LA	SOL*	FA*	MI
4	9	9	9	9	13	11	9	8	6	4
1	1	¾	¼	1	¾	¼	1	¾	¼	1½

"Hearts of Oak"

Fig. 8.2.

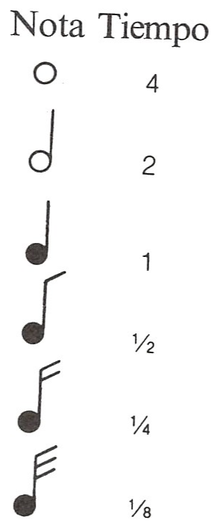


Fig. 8.3. Duración de las notas.

la melodía, tiene ahora dos números asociados a ella: su tono y duración, o tiempo durante el que debe sonar. Esta información se almacena mejor en una sentencia DATA, y después se introduce en dos variables.

Introduzca el programa:

```
1Ø DATA
4,1,9,1,9,.75,9,.25,9,1,13,.75,11,.25,9,1,8,.75,6,.25,4,1.5,99,99
2Ø LET temp=.25
3Ø READ p,t
4Ø IF p=99 THEN STOP
5Ø BEEP t*temp,p
6Ø GOTO 3Ø
```


La sentencia DATA termina con dos valores 99, que sirven para detectar el final de la melodía. La variable "temp" especifica la longitud del intervalo de tiempo fundamental, en este ejemplo un cuarto de segundo, pero usted puede experimentar dando diferentes valores a "temp".

Esperas: PAUSE

La única cosa que no permite el comando BEEP es hacer una pausa para tener un período de silencio. El Spectrum posee otro comando que hará que deje de hacer cualquier cosa durante un período de tiempo especificado. Pero el formato de este nuevo comando es ligeramente distinto:

PAUSE "expresión aritmética"

Esta nueva sentencia hará que el Spectrum deje de hacer cualquier cosa, durante un tiempo igual al resultado de la expresión aritmética. El único inconveniente es que el tiempo se mide en cincuentavos de segundo, y así:

PAUSE 5Ø

logrará que el Spectrum no haga nada durante **un** segundo. Esto no es demasiado problema, sin embargo, porque:

PAUSE t*5Ø

detendrá al computador durante "t" segundos.

Hay otra característica especial del comando PAUSE: si se presiona cualquier tecla, inmediatamente se interrumpe la pausa y el Spectrum continúa ejecutando el programa automáticamente. Si usted escribe:

PAUSE Ø

entonces la pausa no está temporizada y la única forma de que el Spectrum continúe es pulsando alguna tecla.

Aunque el comando PAUSE se ha presentado como un medio de producir silencios musicales, puede emplearse para alterar el tiempo que tarda cualquier parte de un programa en

completarse. El comando PAUSE Ø es muy útil para detener el Spectrum hasta que usted desee seguir adelante.

Algunos efectos de sonido

El inconveniente que existe cuando se usa el Spectrum para producir sonidos es que, mientras el altavoz está emitiendo cualquier ruido, el computador deja de hacer toda clase de cálculos. Por lo tanto, aunque escriba un programa que produzca una excelente imitación del ruido de despegue de un cohete, será bastante difícil añadir cualquier gráfico simultáneo que represente al cohete despegando. Lo mejor que podría hacer es un poquito de ruido, después de mover un poco el cohete, hacer otro poquito de ruido, volver a mover un poco el cohete y así sucesivamente. Un ruido como "woosh" que suene continuamente a la vez que aparece un cohete moviéndose, no es posible de lograr únicamente con BASIC.

A pesar de ello, existen algunos sonidos más, diferentes de los que proporciona BEEP, que pueden ser producidos desde BASIC. Se puede obtener un "click" usando la subrutina:

```
1ØØØ LET a=PEEK 23624/8
1Ø1Ø OUT 254,a-16
1Ø2Ø OUT 254,a
1Ø3Ø RETURN
```

La forma de trabajar de esta rutina no es demasiado importante y exige un conocimiento de los circuitos electrónicos (hardware) del Spectrum; así es que no se preocupe y únicamente, ¡utilícela! La subrutina hará que suene un "click" cada vez que se ejecute el par de instrucciones: OUT 254,a−16 y OUT 254,a. Por ejemplo, pruébela con este programa:

```
1ØØØ LET a=PEEK23624/8
1Ø1Ø FOR i=1 TO 2Ø
1Ø2Ø OUT 254,a-16
1Ø3Ø REM
1Ø4Ø OUT 254,a
1Ø5Ø NEXT i
```

y verá que produce un ruido áspero de tono bajo. La sentencia

REM de la línea 1Ø3Ø se coloca ahí precisamente, para obtener un poco de tiempo entre los "clicks". Si se aumenta el número de sentencias REM, es posible lograr un ruido como de ametralladora.

Se pueden producir otros efectos de sonido, combinando notas de corta duración de frecuencias diferentes con los "clicks" individuales. Los métodos de programar cosas de ese estilo no son difíciles; únicamente se necesitará mucha paciencia e invertir tiempo probando combinaciones diferentes de todas las ideas posibles.

«Ataque al platillo volante»: Función SCREEN$

El juego cuyo listado viene a continuación, hace uso de la mayor parte de los comandos de BASIC que hemos ido presentando en este capítulo y en el capítulo 7.

```
1Ø LET t=15
2Ø LET f=Ø
3Ø LET fx=1Ø
4Ø LET fy=1Ø
5Ø GOSUB 1ØØØ
6Ø FOR x=Ø TO 3Ø
7Ø GOSUB 2ØØØ
8Ø GOSUB 3ØØØ
9Ø PRINT AT 2Ø,x;"[A]"
1ØØ NEXT x
11Ø FOR x=3Ø TO Ø STEP -1
12Ø GOSUB 2ØØØ
13Ø GOSUB 3ØØØ
14Ø PRINT AT 2Ø,x;" [A]"
15Ø NEXT x
16Ø GOTO 6Ø

1ØØØ POKE USR "[A]"    , BIN ØØØØØØØØ
1Ø1Ø POKE USR "[A]"+1, BIN ØØ111111Ø
1Ø2Ø POKE USR "[A]"+2, BIN ØØ111111Ø
1Ø3Ø POKE USR "[A]"+3, BIN Ø11111111
1Ø4Ø INK 2
1Ø5Ø PAPER 5
```

```
1Ø6Ø CLS
1Ø7Ø RETURN

2ØØØ LET f$=INKEY$
2Ø1Ø IF f$="" THEN RETURN
2Ø2Ø BEEP .Ø1,1Ø
2Ø3Ø PRINT AT fy,fx;" "
2Ø4Ø LET fx=x
2Ø5Ø LET fy=21
2Ø6Ø LET f=1
2Ø7Ø RETURN

3ØØØ LET t=t+RND*2-1
3Ø1Ø IF t<5 THEN LET t=t+1
3Ø2Ø IF t>25 THEN LET t=t-1
3Ø3Ø PRINT AT 5,t;" *** "
3Ø4Ø IF f=Ø THEN PAUSE 5:RETURN
3Ø5Ø PRINT AT fy,fx;" "
3Ø6Ø LET fy=fy-1
3Ø7Ø PRINT AT fy,fx;"↑"
3Ø8Ø BEEP .ØØ1,5Ø-fy*2
3Ø9Ø IF fy<4 THEN LET f=Ø:PRINT AT fy,fx;" ":RE-
     TURN
31ØØ IF SCREEN$(fy-1,fx)<>"*" THEN RETURN
311Ø GOSUB 4ØØØ
312Ø RETURN

4ØØØ PRINT AT 5,t+1;FLASH 1;" *** "
4Ø1Ø BEEP .Ø1,4Ø
4Ø2Ø LET a=PEEK 23624/8
4Ø3Ø FOR i=1 TO 2Ø
4Ø4Ø OUT 254,a-16
4Ø5Ø OUT 254,a
4Ø6Ø NEXT i
4Ø7Ø RETURN
```

El juego es relativamente sencillo de dominar. Un "platillo volante" extraterrestre, en forma de tres asteriscos, se mueve nerviosamente por toda la pantalla. Una nave, cuyo propósito es atacar al invasor, se mueve rápidamente hacia delante y hacia atrás en la parte baja de la pantalla. Se puede lanzar un proyectil en cualquier momento desde la nave de combate, pulsando

cualquier tecla. Una vez que el proyectil se ha disparado, se mueve hacia arriba de la pantalla acompañado por un sonido silbante, cuyo tono va en aumento, hasta que alcanza el platillo y se produce una explosión, o falla el blanco. Si se pulsa cualquier tecla durante el vuelo del proyectil, se disparará uno nuevo desde la nave, y el antiguo será borrado de la pantalla.

El programa está escrito como un conjunto de pequeñas subrutinas y no es particularmente difícil de entender. Es más sencillo seguir la parte principal del programa después de haber dado una descripción de cada subrutina, y eso es lo que haremos. La subrutina 1ØØØ-1Ø7Ø define los caracteres gráficos del usuario que forman la nave de combate (líneas 1ØØØ-1Ø3Ø) y establece los colores globales de tinta y de papel (líneas 1Ø4Ø-1Ø7Ø). La subrutina 2ØØØ-2Ø7Ø explora el teclado para ver si se ha pulsado alguna tecla y entonces disparar el proyectil. Si no se ha pulsado ninguna tecla, el control se devuelve a la parte principal del programa (línea 2Ø1Ø). Si, en cambio, se ha pulsado alguna tecla, cualquier proyectil previo se borra de la pantalla imprimiendo un blanco (línea 2Ø3Ø) en la posición actual de dicho proyectil, que está almacenada en "fy" y "fx". Después de eso, en la posición actual del proyectil se almacena la posición actual de la nave de combate (líneas 2Ø4Ø-2Ø5Ø) y la variable "f" se pone a 1 (línea 2Ø6Ø) para indicar que se ha disparado un proyectil y está en vuelo. La subrutina 3ØØØ-312Ø, mueve el platillo una distancia aleatoria hacia la derecha o hacia la izquierda, dibuja el proyectil, si hay uno en vuelo, y comprueba si ha hecho blanco en el platillo. Las líneas 3ØØØ-3Ø3Ø son responsables de mover el platillo. Fíjese que también comprueba la posición de éste para evitar moverlo fuera de la pantalla, en las líneas 3Ø1Ø y 3Ø2Ø. El dibujo del platillo, en la línea 3Ø3Ø, sirve además para borrar de la pantalla el antiguo platillo, puesto que en la cadena de asteriscos que lo representa se han incluido blancos a ambos lados. Las líneas 3Ø4Ø-3Ø8Ø se ocupan de mover el proyectil. La línea 3Ø4Ø comprueba si existe algún misil en vuelo (es decir, f=1). Si no hay ninguno, el control se devuelve al programa principal.

El comando PAUSE se incluye para hacer que la nave de combate se mueva a la misma velocidad, aunque no haya ningún misil en vuelo. Las líneas 3Ø5Ø-3Ø8Ø mueven el misil una línea hacia arriba de la pantalla. La línea 3Ø5Ø borra el anterior proyectil y las 3Ø7Ø hace reaparecer el nuevo, en la posición correcta. La línea 3Ø8Ø provoca un sonido que va aumentando

de tono según asciende el proyectil. Las líneas 3Ø9Ø-311Ø comprueban si el misil ha alcanzado el blanco o ha fallado éste. La línea 3Ø9Ø mira a ver si el proyectil tiene una posición tal que ha sobrepasado el platillo y está a punto de salir fuera de los límites de la pantalla. Si eso ocurre, se imprime un blanco para eliminarle, y la variable "f" se hace igual a Ø para indicar que no hay proyectil en vuelo. La línea 31ØØ emplea la función SCREEN$ para saber qué carácter hay en la posición de pantalla a la que el misil se va a mover. La misión de SCREEN$ no se ha comentado hasta ahora, pero es muy sencillo comprender cómo trabaja. La función SCREEN$ (y,x) devuelve el carácter que hay en la columna "x", línea "y" de la pantalla. Se emplea en la línea 31ØØ para averiguar si el carácter que está justo encima del proyectil es un asterisco. En el caso de ser así, el misil está a punto de hacer blanco en el platillo, y se llama a la subrutina de explosión.

La subrutina 4ØØØ-4Ø7Ø es la rutina de explosión. La línea 4ØØØ imprime el platillo parpadeando para simular la explosión de éste. Las líneas 4Ø2Ø-4Ø6Ø provocan el ruido "áspero", descrito en la sección anterior, para imitar el sonido de una explosión real.

Una vez que hemos descrito todas las subrutinas, el funcionamiento de la parte principal del programa es fácil de entender. Las líneas 1Ø-4Ø almacenan los valores iniciales en algunas de las variables principales: "t" es la posición horizontal del platillo; "f" se usa para indicar cuándo hay un misil disparado que está en vuelo, y "fx", "fy" son las coordenadas del misil. Después, se llama a la subrutina 1ØØØ para especificar los gráficos de usuario y los colores empleados. El trabajo primordial del programa lo llevan a cabo los dos bucles FOR: 6Ø-1ØØ y 11Ø-15Ø.

El primer bucle FOR mueve la nave de combate una posición hacia la derecha cada vez que se ejecuta. La subrutina 2ØØØ se llama cada vez que la nave de combate se mueve, para investigar si hay que disparar un proyectil, y la subrutina 3ØØØ se llama para mover el proyectil y el platillo.

El segundo bucle FOR mueve la nave de combate hacia la izquierda, pero en todo lo demás es idéntico al primer bucle FOR.

Con esto termina la descripción de este pequeño programa de juego. Si usted lo estudia hasta que crea que lo entiende, la mejor forma de averiguar si es así consiste en intentar modificarlo. El juego habría sido mucho más espectacular añadiendo unas

pocas características muy simples. Se podría, por ejemplo, añadir una rutina para que guardase una cuenta del número de platillos alcanzados, o dar a la nave de combate un número limitado de movimientos, antes de que el platillo volante dispare un proyectil contra ella.

Intente probar estas sugerencias y sus propias ideas. Después de todo, ¡la única forma de aprender a programar es programando!

Capítulo 9

Gráficos de alta resolución

Es muy posible que la presencia de los gráficos de alta resolución sea una de las cosas que más influyeron en su decisión de comprar un Spectrum. Sin embargo (puede que los capítulos 7 y 8 ya le hayan convencido de ello), los gráficos de alta resolución no son realmente necesarios para la mayor parte de las aplicaciones. El dibujar cosas en alta resolución normalmente lleva más tiempo y en la mayoría de las ocasiones estará usted limitado al uso de los colores. Una vez colocados los gráficos de alta resolución en el contexto apropiado, debe decirse que los comandos que manejan esta características del Spectrum no son difíciles de entender y de usar, después que haya dominado las ideas principales sobre los gráficos de baja resolución, y que algunos de los efectos que pueden lograrse son realmente muy buenos.

La pantalla de alta resolución

El Spectrum usa el mismo método para mostrar en la pantalla los gráficos tanto de alta como de baja resolución. Este no es el caso de otros computadores y, ya que ello significa que se pueden mezclar los textos con los gráficos de alta resolución, ¡es una ventaja a favor del Spectrum!

La forma en que funciona la alta resolución, es fácil de comprender en términos de las cuadrículas de ocho por ocho puntos, que componen cada carácter en la pantalla. Los comandos de baja resolución pueden alterar únicamente el bloque completo de ocho por ocho; pero los comandos de alta resolución pueden cambiar un único punto de cualquier posición de carácter de la pantalla. Observe que esto significa que **todas** las reglas para imprimir un punto de un carácter en la pantalla permanecen igual. Por ejemplo, el color que tendrá un punto,

dependerá de si éste es de tinta o de papel y de lo que haya sido establecido por los comandos INK y PAPER. Los gráficos de alta resolución únicamente nos proporcionan nuevos medios de poder cambiar los puntos de una posición de impresión, no el color ni ninguno de los restantes atributos de dicha posición.

Evidentemente, si se pueden utilizar los nuevos comandos para cambiar puntos elementales, debe de existir alguna forma de especificar tales puntos. Teóricamente se podría lograr esto con la numeración existente para las posiciones de carácter, y referirnos a los puntos individuales como, por poner un caso, el tercer punto de un carácter en particular. Pero resulta que es mucho más simple tener una numeración totalmente nueva para referirnos a los puntos. Puesto que hay treinta y dos caracteres en una línea y cada carácter posee ocho puntos de ancho, existen un total de 32*8, es decir, 256 puntos en una línea. Como, por otro lado, la pantalla está formada por veintidós líneas y cada una de ellas tiene ocho puntos de alto, son 22*8, o sea 176 puntos verticalmente. Entonces, la pantalla de alta resolución se compone de 256 puntos horizontales y 176 puntos verticales, y podemos referirnos a cualquiera de ellos especificando la fila y la columna en que se encuentra. Las columnas están numeradas desde cero, empezando en el extremo izquierdo de la pantalla, y, en consecuencia, el número de columna o coordenada x, como se denomina, comprende el intervalo de Ø a 255. Las filas están numeradas desde cero, comenzando en la parte inferior de la pantalla. Así pues, el número de fila o coordenada y, que es su nombre alternativo, comprende un intervalo de Ø a 175 (observe que los números de fila van de la parte inferior hasta la parte superior, mientras que la numeración de las líneas de caracteres comienza en la parte superior de la pantalla). Cualquier punto de la pantalla se puede designar dando dos números: su coordenada x, y su coordenada y. Normalmente, se escriben esos dos números entre paréntesis, colocando primero la coordenada x. Es decir: (Ø,Ø) es el extremo inferior izquierdo, y (255,175) es el extremo superior derecho de la pantalla. Después de un poco de práctica, el empleo de las coordenadas x é y llegará a ser su segunda naturaleza.

Los comandos gráficos: PLOT, DRAW y CIRCLE

Los comandos de alta resolución son mucho más sencillos de entender cuando están trabajando con sólo dos colores, y, por

ello, empezaremos por ahí. A lo largo de toda la discusión que sigue se supondrá que usted no ha cambiado la situación inicial de la pantalla de su Spectrum, papel de color blanco y tinta de color negro, o que ha especificado dos colores que contrasten razonablemente usando los comandos INK y PAPER.

El comando más simple de alta resolución es:

PLOT "expresión aritmética 1", "expresión aritmética 2"

Su efecto es cambiar el punto que está en la coordenada x, dada por "expresión aritmética 1" y la coordenada y, dada por "expresión aritmética 2", a punto de tinta. Pruebe el programa siguiente para descubrir cómo funciona PLOT e investigar la forma de trabajar de las coordenadas x,y:

```
1Ø INPUT "x=";x;" y=";y
2Ø PLOT x,y
3Ø GOTO 1Ø
```

Si introduce un valor para x é y, que quede fuera de los límites de la pantalla, obtendrá un error: "Integer out of range". Para una demostración automática puede emplear:

```
1Ø LET x=INT (RND*256)
2Ø LET y=INT(RND*176)
3Ø PLOT x,y
4Ø GOTO 1Ø
```

El comando más útil de los que existen en alta resolución es:

DRAW "expresión aritmética 1", "expresión aritmética 2"

que produce una línea recta. La posición de comienzo de la línea es el punto donde el último PLOT o DRAW terminó, y su final es "expresión aritmética 1", puntos hacia la derecha, y "expresión aritmética 2", puntos hacia arriba. Por ejemplo, si el último comando PLOT fue PLOT Ø,Ø, el comando:

DRAW 1ØØ,1ØØ

producirá una línea desde (Ø,Ø) a (1ØØ,1ØØ). Pero si el último PLOT fue PLOT 5Ø,5Ø, la línea habría comenzado en (5Ø,5Ø) y

habría finalizado en (15Ø,15Ø). Es importante resaltar que DRAW usa las coordenadas de forma completamente diferente de como las emplea PLOT.

Cuando se trabaja con alta resolución, se puede imaginar que hay un **cursor gráfico** que se mueve alrededor de la pantalla, con la posición actual de gráficos de forma muy similar a lo que sucedía con el cursor de textos, que se movía por la pantalla, indicando la posición actual de impresión. El comando DRAW x,y mueve el cursor gráfico x unidades en dirección horizontal, é y unidades en dirección vertical, y luego traza una línea recta desde la posición inicial del cursor hasta la nueva posición. Los comandos como RUN, CLEAR, CLS y NEW, vuelven a colocar el cursor gráfico en el punto (Ø,Ø). Después de eso, el cursor gráfico se mueve por la pantalla cuando se ejecuta algún comando gráfico. La forma más clara de ver que DRAW x, y, es diferente de PLOT x,y, puede ser un comando como:

DRAW−1Ø,1Ø

que traslada el cursor de gráficos 1Ø unidades a la izquierda y 1Ø unidades hacia arriba. ¡Las coordenadas negativas no están permitidas en PLOT!

El comando DRAW es a menudo muy conveniente, pero puede ser difícil trazar una línea entre dos puntos dados. No obstante, la combinación siguiente dibujará una recta entre el punto (x1, y1) y (x2, y2):

PLOT x1,y1: DRAW x2−x1, y2−y1

Para poner de manifiesto la clase de cosas que puede hacer DRAW, introduzca este pequeño programa:

```
1Ø FOR i=1 TO 175 STEP 4
2Ø PLOT Ø,i
3Ø DRAW 255−i,−i
4Ø PLOT i,Ø
5Ø DRAW 255−i,i
6Ø PLOT Ø,i−75
7Ø DRAW 255−i,i
8Ø PLOT i,175
9Ø DRAW 255−i,−i
1ØØ NEXT i
```

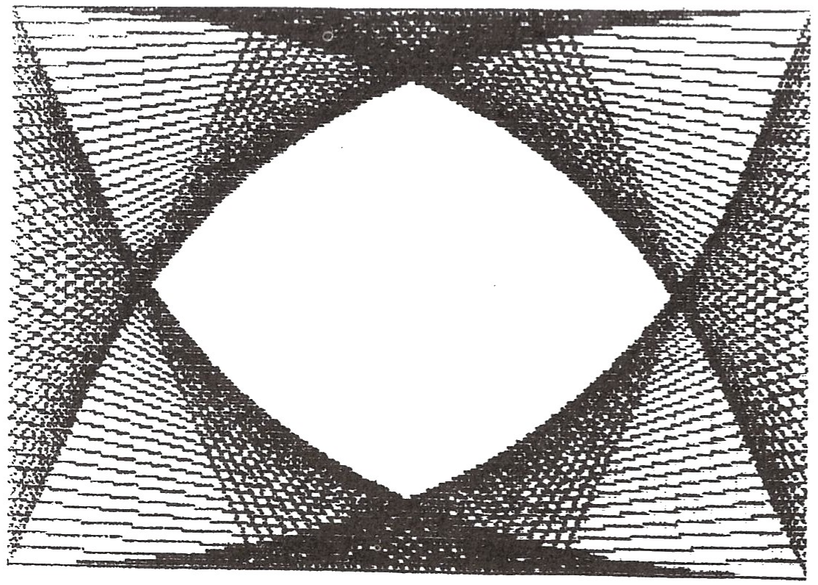


Fig. 9.1. Dibujo de alta resolución.

La salida de este programa se reproduce en la figura 9.1. Puede comprobar que se obtienen diferentes efectos variando el valor de STEP, en la línea 1Ø. Aunque aún no hemos agotado completamente las posibilidades del comando DRAW, su característica adicional se comprenderá mejor después de haber explicado el comando gráfico CIRCLE, último de dichos comandos.

El comando:

CIRCLE	"expresion aritmetica 1",	"expresion aritmetica 2",	"expresion aritmetica 3"

dibujará un círculo centrado en el punto (x,y), donde x é y son los valores de las dos expresiones aritméticas que figuran primero en el comando, y con un radio determinado por el valor de "expresión aritmética 3".

Por ejemplo:

```
CIRCLE 1ØØ,5Ø,4Ø
```

traza un círculo centrado en el punto (1ØØ,5Ø) de radio 4Ø. Como una muestra del comando CIRCLE, vamos a ver el siguiente programa, que dibuja círculos aleatorios y cuya salida aparece en la figura 9.2.

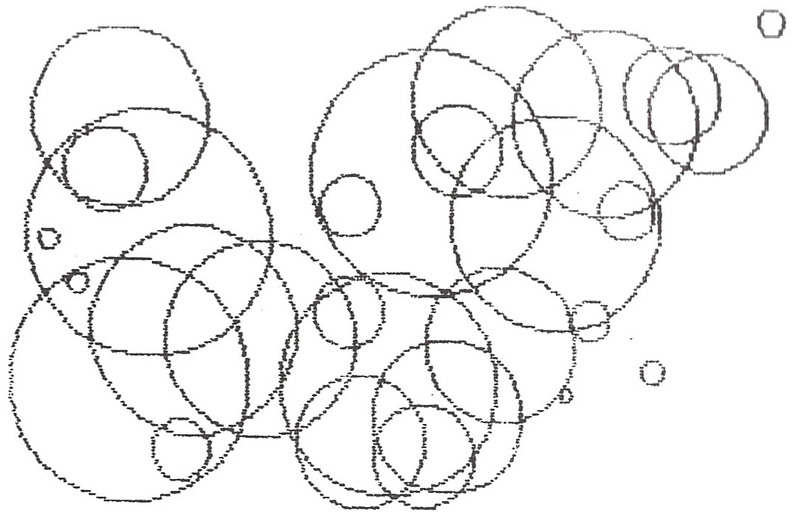


Fig. 9.2. Dibujo de círculos al azar.

```
1Ø LET r=RND*5Ø
2Ø LET x=r+RND*(255-2*r)
3Ø LET y=r+RND*(175-2*r)
4Ø CIRCLE x,y,r
5Ø GOTO 1Ø
```

Cuando dibuje círculos aleatoriamente, tendrá que tener cuidado de que ninguno de ellos sobrepase los límites de la pantalla. En este programa, esta circunstancia se evita con la inclusión de −2*r en las líneas 2Ø y 3Ø.

Ahora que conocemos el comando CIRCLE, podemos ocuparnos de la característica adicional del comando DRAW de la que hablamos antes. Además de poder trazar líneas rectas entre dos puntos, DRAW puede usarse para dibujar arcos de circunferencia entre dos puntos.

La forma más general del comando DRAW es:

DRAW	"expresion aritmetica 1",	"expresion aritmetica 2",	"expresion aritmetica 3"

El significado de las dos primeras expresiones ya lo conocemos y permanece inalterado por la existencia de la tercera expresión. Sin embargo, el tercer parámetro especifica un ángulo, empleado para determinar qué longitud de arco de circunferencia será dibujado. Para entender cómo puede un ángulo

especificar qué porción de círculo se va a dibujar, todo lo que tiene que hacer es imaginar que está en el centro del círculo. Si extiende los brazos formando un ángulo determinado, la parte de círculo que está entre sus brazos será la parte de círculo que trazará DRAW. Por ejemplo, 180 grados especifica media circunferencia, y 90 grados un cuarto de ésta. La única complicación es que el Spectrum mide los ángulos en radianes en vez de grados. Para pasar de grados a radianes, simplemente hay que multiplicar por 180 y dividir por PI(π). Según esto, un ángulo de PI radianes equivale a un semicírculo, un ángulo de PI/2 determina un cuarto de círculo, y así sucesivamente. Como ejemplo del comando DRAW, pruebe:

```
1Ø PLOT 1ØØ,1ØØ: DRAW 5Ø,5Ø,PI/2
```

El primer PLOT mueve el cursor gráfico a (1ØØ,1ØØ), y después el comando DRAW dibuja un cuarto de círculo comenzando en (1ØØ,1ØØ) y terminando en (15Ø,15Ø). Observe que la porción de círculo que va a dibujar DRAW la determina exclusivamente el tercer parámetro. El tamaño y la orientación de la misma depende de la posición actual del cursor de gráficos y de los dos primeros parámetros.

Es probable que ya se haya dado cuenta de que, a diferencia del caso de una línea recta, hay dos arcos de círculo posibles entre dos puntos. Por ejemplo, si se imagina los dos puntos sobre una línea horizontal, puede dibujar un semicírculo desde el punto de la izquierda hasta el de la derecha, en el mismo sentido de las agujas del reloj o en sentido contrario. El primer caso produciría un semicírculo por encima de la línea horizontal, y en el segundo caso en sentido contrario a las manecillas del reloj; el semicírculo obtenido quedaría debajo de dicha línea.

Esta descripción en términos del movimiento de las agujas del reloj, es precisamente la que utiliza el Spectrum para resolver el problema de cuál de los dos posibles arcos debe dibujar. Los ángulos positivos harán que se dibuje un arco entre los dos puntos en sentido contrario a las agujas del reloj, y los ángulos negativos en el mismo sentido de las agujas del reloj.

Para mostrar el uso de un comando DRAW completo, introduzca el siguiente programa, cuya salida sólo puede definirse como una cuerda enredada (ver figura 9.3):

```
1Ø PLOT 1ØØ,1ØØ
2Ø LET x=25-RND*5Ø
```



```
3Ø LET y=25-RND*5Ø
4Ø LET p=PI*RND
5Ø DRAW x,y,p
6Ø GOTO 2Ø
```

Fig. 9.3. Programa de cuerda enredada.

La línea 1Ø sitúa el cursor de gráficos aproximadamente en el centro de la pantalla, y las líneas 2Ø y 3Ø asignan valores aleatorios a los tres parámetros de DRAW. El ángulo aleatorio se convierte en la línea 4Ø a una fracción de PI, de forma que el comando DRAW de la línea 5Ø dibuja porciones de semicírculos. Si deja que el programa se ejecute durante un tiempo suficientemente largo, se detendrá con un mensaje de error, cuando uno de los arcos intente, finalmente, sobrepasar los límites de la pantalla. En un caso general, resulta bastante difícil detectar si los arcos que dibuja DRAW caerán o no fuera del área de pantalla, ¡hasta que realmente ocurre, claro!

Colores en alta resolución

Se pueden emplear todos los comandos de color y de atributos como INK, PAPER, OVER, etc., que presentamos en el capítulo 7, para controlar la forma en que los puntos de alta resolución aparecen en pantalla. Por ejemplo, si quiere producir puntos azules con un fondo amarillo, todo lo que debe hacer es ejecutar INK 1: PAPER 6, antes de cualquier comando PLOT, CIRCLE o DRAW. Se pueden usar los comandos de color, incluso dentro de los comandos gráficos, como se hacía en el caso del comando PRINT. En ese caso, el efecto de cualquier

comando de color es temporal y únicamente afectará a los puntos que produzca el comando en el que están incluidos. La única cosa que nunca debe olvidar es que los colores y los atributos (salvo el caso de OVER e INVERSE) se aplican a las posiciones de impresión de carácter completas. Es decir, si se dibuja un punto único con PLOT INK 4; 1ØØ,1ØØ, entonces **todos** los puntos de tinta que haya en la misma posición de carácter de (1ØØ,1ØØ) cambiarán al color 4 (verde). La demostración más sorprendente de cómo afectan los comandos INK y PAPER a las posiciones de carácter completas, se obtiene dibujando una línea con DRAW con un color de papel diferente al del resto de la pantalla:

```
1Ø PAPER 6
2Ø CLS
3Ø DRAW PAPER 1; INK 4; 1ØØ,1ØØ
```

Estas tres líneas producen un efecto bastante extraño en la pantalla, porque cada punto dibujado cambia el color de todos los puntos de papel de la posición de carácter a la que pertenece dicho punto, al color 1 (azul). Si hubiese cualquier punto de tinta de otro color en la misma posición de carácter, su color pasaría a ser verde.

El hecho de que los colores (incluyendo BRIGHT y FLASH) afectan a las posiciones de impresión completas, impone una severa restricción a la forma de usar colores en alta resolución. Si dibuja dos líneas mediante DRAW, con colores de tinta diferentes, no habrá ningún problema, a menos que ambas lleguen a coincidir en una misma posición de impresión de la pantalla. Cuando ocurre esto último, la primera línea dibujada cambiará de color en la posición de carácter donde coinciden ambas, al color de la segunda línea. La regla es que sólo pueden existir dos colores en una única posición de impresión de carácter en la pantalla: el color de la tinta y el del papel. Mientras las líneas, círculos..., etc., de alta resolución ocupen zonas diferentes de la pantalla, cada una podrá tener cualquiera de los ocho colores; pero si una de ellas comparte una misma posición de carácter con otra, en esa posición ambas deben tener el mismo color de tinta y el mismo de papel. Para ciertas aplicaciones, esta restricción no tiene importancia, puesto que diferentes líneas de distintos colores ocuparán normalmente zonas separadas de pantalla. Sin embargo, si eso no es posible, no hay nada que se

pueda hacer para superar esta limitación de los gráficos de alta resolución del Spectrum. Para evitar esa clase de problemas, lo mejor que puede hacerse es considerar la pantalla de gráficos de alta resolución como una pantalla de dos colores.

Borrado de puntos: OVER e INVERSE

Aunque los comandos de color INK, PAPER, FLASH y BRIGHT, afectan a una cuadrícula completa de puntos de las que componen un carácter, los comandos OVER e INVERSE únicamente afectan la forma en que cada punto elemental se muestra en la pantalla. Esto significa que OVER e INVERSE son verdaderos comandos de alta resolución, que se pueden usar dentro de los comandos PLOT, DRAW y CIRCLE sin afectar a nada de lo que ya estuviese en la pantalla. Por ejemplo, INVERSE 1 hace que se produzcan puntos de papel en vez de puntos de tinta.

Así el programa:

```
1Ø CIRCLE INVERSE Ø;1ØØ,1ØØ,5Ø
2Ø CIRCLE INVERSE 1;1ØØ,1ØØ,5Ø
3Ø GOTO 1Ø
```

primero dibuja un círculo con puntos de tinta y luego lo borra colocando puntos de papel en el sitio de cada punto de tinta anterior.

El comando OVER es más complicado de entender, porque sus efectos dependen de lo que haya ya en la pantalla, en la posición donde se va a dibujar el punto. Recordando las reglas que dimos para saber cómo funciona OVER en el capítulo 7, podrá ver que una línea de puntos dibujada en alta resolución a continuación de un OVER 1, aparecerá como una línea de puntos de tinta, si estos puntos se van dibujando sobre puntos de papel y como una línea de puntos de papel, si los puntos se van dibujando sobre puntos de tinta que ya hubiese en la pantalla. Esto es muy útil en la práctica, pues nos proporciona un medio de dibujar y borrar puntos de alta resolución, sin perder ninguno de los puntos que estuviesen ya presentes en la pantalla. Para ver esto en acción, pruebe:

```
1Ø CIRCLE 1ØØ,1ØØ,5Ø
2Ø PLOT Ø,1ØØ
```

```
30 DRAW OVER 1;255,0
40 PAUSE 20
50 GOTO 20
```

El primer comando DRAW produce una línea de puntos de tinta que atraviesa el círculo dibujado por la línea 10, salvo donde la línea corta el círculo, donde, por las reglas que acabamos de ver, se producen puntos de papel. Después de la pausa que proporciona la línea 40 para que pueda observar la figura, los comandos PLOT y DRAW se ejecutan de nuevo. Esta vez, los puntos de tinta de la línea se transforman en papel y la línea desaparece, salvo los puntos donde la línea cruzaba el círculo, que eran de papel y ahora vuelven a tener su valor inicial de tinta. Esta aparición y desaparición continuará hasta que pulse BREAK desde el teclado. Por medio de esto, si se desean producir líneas o círculos temporalmente, dibújelos usando OVER 1 y después elimínelos empleando OVER 1 de nuevo.

Es mejor resumir el uso de OVER e INVERSE en los comandos de alta resolución, y para ello usamos PLOT como ejemplo:

PLOT	dibuja un punto de tinta
PLOT INVERSE 1	dibuja un punto de papel
PLOT OVER 1	cambia el color de un punto que ya estuviese en la pantalla, al color opuesto, o sea, tinta a papel y viceversa
PLOT INVERSE 1; OVER 1	no tiene ningún efecto, excepto mover el cursor de gráficos

Averiguando qué hay en la pantalla: POINT

De la misma forma que SCREEN$ podía emplearse para descubrir el carácter que había en cualquier posición de impresión de la pantalla, la función POINT puede utilizarse para averiguar qué clase de punto hay en la pantalla, en cualquier posición de alta resolución.

La forma general de la función POINT es:

POINT ("expresion aritmetica 1", "expresion aritmetica 2")

donde "expresion aritmetica 1" es la coordenada x, y "expresión aritmetica 2" es la coordenada y. A diferencia de la mayor parte de las funciones del Spectrum, ésta **exige** los paréntesis que encierran sus dos parámetros. El valor que devuelve POINT es 1, si en las coordenadas especificadas hay un punto de tinta, y un 0, si lo que existe es un punto de papel. Las aplicaciones de esta función son similares a las que tenía SCREEN$, de las cuales era un ejemplo su empleo en el programa del platillo volante del capítulo 8.

Uso de los gráficos de alta resolución

A pesar de que sólo existen unos pocos comandos para gráficos de alta resolución, puede resultar difícil ver cómo pueden emplearse para producir buenos dibujos. Como siempre, el mejor modo de aprender es ver algunos ejemplos y después intentar escribir sus propios programas.

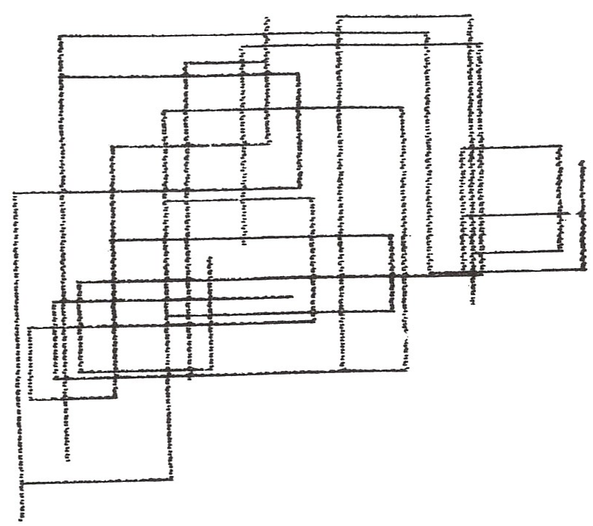


Fig. 9.4. Programa de dibujo a mano alzada.

El primer ejemplo usa la función INKEY$ para controlar la posición de un punto en la pantalla. Pulsando las teclas de las flechas adecuadas, usted puede dibujar en la pantalla:

```
10 LET x=127
20 LET y=80
30 LET a$=INKEY$
```

```
40 IF a$="5" THEN LET x=x-1
50 IF a$="8" THEN LET x=x+1
60 IF a$="6" THEN LET y=y-1
70 IF a$="7" THEN LET y=y+1
80 PLOT x,y
90 GOTO 30
```

En la figura 9.4 puede ver una muestra de los resultados del programa. Podría intentar añadir algunas mejoras "de su cosecha", tales como movimientos en diagonal y la capacidad de moverse de un punto a otro de la pantalla sin dibujar ninguna línea. El segundo ejemplo se basa en un conjunto de figuras descubiertas por el físico francés del siglo XIX, Lissajous, y que se denominan en su honor: "Figuras de Lissajous":

```
10 LET t=0
20 LET t=t+0.1
30 LET x=50*(1+SIN(1.1*t))
40 LET y=50*(1+COS t)
50 PLOT x+50,y+50
60 GOTO 20
```

En la figura 9.5 puede ver la salida del programa. Se puede obtener toda una serie de tales figuras, cambiando el valor 1.1 que aparece en la línea 30 por otros diferentes.

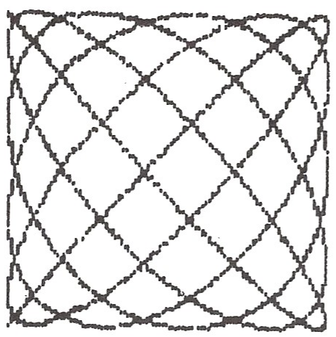


Fig. 9.5. Figura de Lissajous.

El ejemplo final de este capítulo es un programa que dibuja la gráfica de SIN(x)/x. Esta función tiene una forma particularmente interesante, como podrá observar en la figura 9.6. Una dificultad que hay que tener muy en cuenta, es que SIN(x)/x es

imposible de calcular cuándo x vale 0 (no está definida en x=0). Por ello, hay que tener la precaución de dejar ese punto fuera del gráfico:

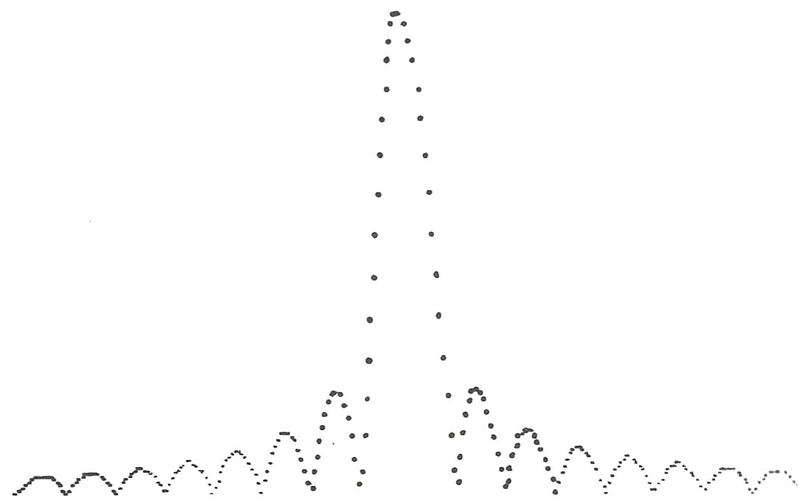


Fig. 9.6. Gráfico de SIN(x)/x.

```
10 FOR i=1 TO 255
20 IF i=127 THEN GOTO 60
30 LET x=(i-127)/5
40 LET y=150*SIN(x)/x
50 PLOT i,y
60 NEXT i
```

Cuando use gráficos en sus propios programas, la mejor estrategia es limitar el empleo de los gráficos de alta resolución a los casos en que éstos lleven a cabo una función esencial. En otras palabras, es aconsejable partir siempre de los gráficos de baja resolución, estando atentos a las situaciones donde los gráficos de alta resolución harán mejor el trabajo.

Capítulo 10
Lógica y otros temas

A estas alturas, usted podría escribir cualquier programa que desee, empleando el BASIC y las demás informaciones que hemos presentado en los capítulos anteriores. No obstante, hay algunas posibilidades del Spectrum que, aunque no son indispensables, harán que las cosas sean más fáciles o irán más allá de lo que puede hacerse únicamente con BASIC. Este capítulo agrupa estas extensiones y explica un poco su funcionamiento.

El primer tema que vamos a tratar tiene un título más bien intimidante: **la lógica.** En él se presentan los comandos AND, OR y NOT. En segundo lugar, veremos los comandos PEEK, POKE, IN, OUT, USR y CLEAR, que afectan al funcionamiento interno del Spectrum. Después de eso, consideraremos el comando ATTR, que puede utilizarse para descubrir el color de cualquier punto de la pantalla y, finalmente, comentaremos una de las sentencias BASIC más simple: REM.

Sería un error creer que este conjunto de temas carecen de importancia, sólo porque no han figurado hasta ahora en el libro. Se presentan en último lugar, pero no con menor atención.

La lógica y las expresiones condicionales

En las conversaciones cotidianas se dicen a menudo frases como: "¿compraste manzanas y naranjas?", "¿prefieres té o café?". El empleo de palabras como "y", "o" es tan común, que muy pocas veces nos paramos a pensar en ellas. Obviamente, sería una gran ventaja poder utilizar, en las expresiones condicionales de BASIC, cosas como "**y**" y "**o**" **(and, or,** en inglés). Afortunadamente, la mayoría de las versiones de BASIC permiten incorporar estos conceptos cotidianos, y en el BASIC del Spectrum se pueden escribir expresiones tales como:


```
a<Ø AND b=3
a<Ø OR b=3
```

El significado de cada una de esas expresiones es muy similar al que poseen las palabras "y" (AND), "o" (OR) en el lenguaje corriente. La primera de las expresiones anteriores es verdadera, si las condiciones "a<Ø", "b=3" son verdaderas **ambas;** la segunda expresión será verdadera, si cualquiera de las condiciones "a<Ø", "b=3" es verdadera.

Del mismo modo que AND y OR, el Spectrum permite emplear NOT (no), que simplemente cambia el valor de una expresión condicional de verdadero a falso y viceversa. Por ejemplo, "3=2" es falso, pero "NOT 3=2" es verdadero.

Se pueden combinar AND, OR y NOT con cualquiera de las condiciones que conocíamos del capítulo 4, para formar expresiones más complejas que tomarán siempre uno de dos valores: **verdadero** o **falso** (como ya sabe desde el capítulo 4, en las expresiones del Spectrum, verdadero se representa con un "1", y falso con un "Ø"). Las expresiones condicionales que incluyen AND, OR o NOT se denominan normalmente: **expresiones lógicas,** y ahora podemos reescribir la definición de la sentencia IF de esta forma:

```
IF "expresion logica" THEN "sentencia BASIC"
```

Si, por ejemplo, quiere comprobar que una coordenada x no va a quedar fuera de los límites de la pantalla, podría hacerlo así:

```
IF x<Ø OR x>255 THEN...
```

en lugar de escribir dos sentencias IF, que se habrían necesitado si no hubiésemos utilizado OR.

Construir expresiones lógicas para comprobar condiciones globales es, generalmente, inmediato. Sin embargo, existen ciertas pegas que confunden incluso a los expertos. Si desea traducir la frase "b igual a c y d", deberá repetir la condición "=". En otras palabras, debe escribir:

```
b=c AND b=d
```

y no poner, en cambio:

```
b=c AND d
```

que daría un resultado dependiente de si el valor de "d" es Ø o 1. También debería tener cuidado cuando emplee NOT. Por ejemplo:

NOT (a=b AND a=c)

no es lo mismo que:

NOT (a=b) AND NOT (a=c)

Para comprobar esto último, intente averiguar el valor de ambas expresiones para unos cuantos valores de "a", "b" y "c". La moraleja es que siempre debe evitar emplear expresiones lógicas, sin pensar antes qué significan exactamente.

Para cerrar este tema de expresiones lógicas, puede ser interesante presentar la idea de **tabla de verdad.** Consideremos la expresión lógica:

a AND b

donde "a" y "b" son variables cuyo valor es o "1" para verdadero, o "Ø", para falso. Se puede construir una tabla que recoja el resultado de la expresión para todos los valores de "a" y "b", como sigue:

a	b	a AND b
Ø	Ø	Ø
Ø	1	Ø
1	Ø	Ø
1	1	1

Una tabla como ésta se denomina **«tabla de verdad»,** porque contiene las condiciones para las que la expresión es verdadera o falsa. Las tablas de verdad, se pueden construir para cualquier expresión lógica y ésta es una manera de comprobar si se entiende o no el funcionamiento de dichas expresiones.

Por ejemplo OR y NOT tienen las tablas siguientes:

a	b	a OR b	NOT a
Ø		Ø	Ø	1
Ø		1	1	1
1		Ø	1	Ø
1		1	1	Ø

El OR que hemos estado usando hasta ahora no es totalmente equivalente a la palabra "o" del lenguaje corriente. Normalmente, cuando empleamos "o" en nuestros diálogos de la vida corriente, significa "una cosa u otra, pero no las dos al mismo tiempo". Ejemplo: "Puedes comer jamón o mermelada", significa que tú puedes comer una de las dos, pero no las dos cosas (normalmente). Sin embargo, el OR (o) lógico significaría en la misma frase, tú puedes comer una cosa, o la otra, o ambas. El OR lógico se suele llamar, más apropiadamente, **«o inclusivo»,** porque incluye la posibilidad de que ocurran ambas cosas.

La palabra "o" corriente, con el sentido explicado antes, se conoce con el nombre de **«o exclusivo»,** puesto que excluye la posibilidad de que ocurran ambas cosas a la vez. Se puede construir una expresión lógica que es equivalente al o exclusivo:

o exclusivo=(NOT(a) AND b) OR (a AND NOT (b))

como puede comprobarse con la tabla de verdad:

a	b	(NOT(a) AND b) OR (a AND NOT (b))
Ø	Ø	Ø
Ø	1	1
1	Ø	1
1	1	Ø

Esta secuencia de Ø (falso) y 1 (verdadero), tal vez le recuerde la regla de los puntos de tinta y de papel del capítulo 7, donde discutimos el comando OVER. De hecho, después de OVER 1 el punto producido en la pantalla es el "o exclusivo" entre el punto que ya había en la pantalla y el nuevo que se quiere dibujar. En cuanto se fije un poco, verá de inmediato cómo la lógica surge inesperadamente en los sitios más extraños.

Dentro del Spectrum: BIN, PEEK, POKE, IN, OUT, USR y CLEAR

Puede resultar sorprendente saber que BASIC incluye una serie de comandos que permiten el acceso al funcionamiento interno del Spectrum. La razón por la que esto podría parecer extraño es que, hasta ahora, todo lo que hemos visto sobre BASIC trataba de evitar al máximo el preocuparse por la forma en que la máquina ejecutaba internamente los comandos. Sin

embargo, existen algunas aplicaciones en las que las instrucciones normales de BASIC son deficientes de alguna forma. Por ejemplo, podrían ser demasiado lentas o quizás podrían no acceder a determinadas características del computador. Para permitir al programa salvar estas dificultades, la mayor parte de las versiones de BASIC incluyen instrucciones mediante las cuales se puede acceder al interior de la máquina.

Ya encontramos la función BIN en el capítulo 7, donde se usó sin ninguna explicación para construir caracteres definidos por el usuario. Aunque, desde el punto de vista del programador de BASIC, el Spectrum efectúa sus cálculos con números decimales, en realidad cualquier clase de cálculo es ejecutado en un sistema de numeración más elemental denominado **binario.** Las personas cuentan en decimal simplemente porque tienen diez dedos. Si únicamente tuviésemos dos dedos, entonces contaríamos de la misma forma en que lo hacen los computadores, en binario. Aunque no es importante para nadie saber mucho acerca de este sistema de numeración, porque el BASIC cuida de convertir los números decimales en binarios y viceversa, sí que lo es si usted quiere utilizar su Spectrum directamente, sin ayuda del BASIC. Incluso en ese caso, sólo necesitará saber que la función BIN tomará un número binario como entrada y lo transformará en un número decimal. Un número binario es un número que contiene únicamente ceros y unos. Eso es todo lo que precisa conocer, porque BASIC y BIN harán el resto. Por ejemplo, pruebe:

```
PRINT BIN "numero binario"
```

que imprimirá el equivalente decimal de cualquier número binario que usted ponga detrás de BIN. Puede comprobar que 11 es 3, 111 es 7 y 11111111 (ocho unos) es 255 (vea como, al contrario que otras funciones del Spectrum, BIN no permite que use expresiones).

Aprendimos al comienzo, que una variable es un área de la memoria del computador con un nombre. A pesar de ello, algunas veces es necesario dejar a un lado esta forma de usar la memoria del ordenador y utilizarla en acceso directo, por medio de PEEK y POKE. PEEK es una función que devolverá el contenido de una posición de memoria y POKE es un comando que cambiará el contenido de una posición de memoria. Es todo tan simple como eso, salvo que necesita saber cómo especificar

qué posición de memoria y qué clase de número puede guardarse en dichas posiciones. El primer problema se soluciona fácilmente, porque el Spectrum, como todos los computadores, numera todas sus posiciones de memoria secuencialmente, empezando desde cero. Así, PEEK(543) nos dará el contenido de la posición número 543. El segundo problema también se resuelve con sencillez, una vez que sepa que una posición de la memoria puede almacenar un número binario de ocho ceros o unos como máximo (ocho cifras binarias) y como BIN 11111111 es equivalente a 255 decimal, éste será el número mayor que se puede tener guardado en una única posición de memoria. Entonces, POKE 1ØØØ,2ØØ almacenará 2ØØ en la posición 1ØØØ de la memoria. En cambio, POKE 1ØØØ,6ØØ dará un error, porque 6ØØ es mayor que 255. En general, para emplear PEEK y POKE tiene que conocer qué hay almacenado en el interior del Spectrum, y eso, muchas veces, no es fácil de averiguar. Por este motivo sería mejor que evitara manejar PEEK y POKE, a menos que esté absolutamente seguro de lo que está haciendo.

A pesar de todo, existen una o dos aplicaciones típicas de ambos comandos, que son interesantes y demuestran, además, la forma en que se usan normalmente PEEK y POKE. El Spectrum tiene un **reloj** escondido en sus "profundidades", que hace "tic-tac" cada cincuentavo de segundo. Para un programador, este reloj tiene la apariencia de tres posiciones de memoria cuyos valores están cambiando continuamente. Las tres posiciones de memoria citadas, trabajan juntas para ampliar el rango de tiempo que puede ser almacenado, más allá de lo que podría contener una única posición elemental. La posición 23672, cuenta los cincuentavos de segundo, y como el mayor número que puede contener una única posición es 255, aquella cuenta 255 cincuentavos de segundo y después vuelve a cero. La segunda posición, la 23673, cuenta las veces que la primera ha pasado a cero de nuevo, o sea, contará en unidades de 256 cincuentavos de segundo. De igual manera, la tercera de las posiciones, que es la 23674, cuenta el número de veces que la anterior ha vuelto de nuevo a cero, y eso significa que cuenta en unidades de 256*256 cincuentavos de segundo. Puede hacer uso de esta información, aprovechando la función PEEK para ver qué hay en cada posición de memoria de las comentadas y convertirlo en un número que represente el tiempo en segundos. Por ejemplo:

```
(65536*PEEK 23674+256*PEEK 23673+PEEK 23672)/5Ø
```

le dará el tiempo en segundos que ha transcurrido desde que se arrancó el reloj, lo que ocurre normalmente al conectar el computador. En un modo similar, puede utilizar POKE para "poner el reloj en hora". Por ejemplo, para ponerlo a cero, emplee:

POKE 23674,Ø: POKE 23673,Ø: POKE 23672,Ø

La mayoría de las aplicaciones de PEEK y POKE son análogas a ésta, en el sentido de que exigen que usted sepa algo sobre el lugar donde el Spectrum guarda alguna información que está normalmente oculta. Otro ejemplo: una vez que sepa que la posición 23692 guarda el número de líneas que serán desplazadas hacia arriba de la pantalla, antes de que aparezca el conocido mensaje de "Scroll? y/n", puede utilizar POKE 23692,255 para asegurarse de que el mensaje citado no saldrá en la pantalla hasta que se hayan realizado 255 desplazamientos de ésta hacia arriba.

De una manera parecida, se pueden emplear IN y OUT para comunicar con cualquiera de los dispositivos externos conectados al Spectrum, tal como la impresora ZX. Sin embargo, a menos que usted tenga una aplicación muy especial, no hay ninguna circunstancia que exija utilizar estas últimas. Puede ver un caso en que se utilizaron ambas, en la corta subrutina que presentamos en el capítulo 8 para producir un "click" elemental. Para el Spectrum, el altavoz interno es un **dispositivo externo,** y por eso se necesitan estos comandos para producir sonidos.

También encontramos ya la función USR cuando hablamos de los caracteres definidos por el usuario. Pero su función allí era muy especial y, en general, la función USR transfiere el control fuera de BASIC a un programa en **código máquina** almacenado en alguna parte de la memoria del Spectrum.

El Código Máquina es un tema completamente nuevo y bastante extenso. Por eso, hasta que aprenda bastante acerca de dicho código, la función USR será de muy poco interés para usted.

La última instrucción que vamos a mencionar en esta sección, está también íntimamente relacionada con el tema del código máquina. El comando CLEAR puede reservar memoria para almacenar programas en código máquina, de forma que el Spectrum no asigne ese espacio de memoria al área de almacenamiento de variables. Debido a su función, tan especializada, no es muy probable que se cruce en su camino muy a menudo.

Averiguando los colores de la pantalla: ATTR

Del mismo modo que POINT y SCREEN$ se podían usar para ver lo que había en la pantalla, la función ATTR puede emplearse para averiguar qué colores se están utilizando para tinta y papel, y si una determinada posición de carácter está parpadeando o posee un brillo extra. El único inconveniente es que ATTR devuelve toda esta información en forma de un único número que debe de ser **decodificado** para saber qué significa. No obstante, es posible dar una lista de expresiones que extraerán cada segmento de información de los que hemos citado:

INT (ATTR (línea, columna)/128)

dará Ø, si la posición está estacionaria, y 1 si dicha posición está en modo parpadeo y:

INT ((ATTR (línea, columna)-INT (ATTR (línea, columna)/128)*128)/64)

será Ø, si la posición de carácter tiene brillo extra, y 1 si no lo tiene.

De igual forma:

INT ((ATTR (línea, columna)-INT (ATTR (línea, columna)/64)*64)/8)

da el código del color de papel, y

ATTR (línea, columna)-INT (ATTR (línea, columna)/8)*8

da el código del color de la tinta en todos los casos de la posición de carácter especificada por: (línea, columna).

La sentencia REM y la buena programación

El comando BASIC, REM es el más simple de todos los que existen, puesto que ¡no hace absolutamente nada! Su única finalidad es permitirle incluir comentarios que no forman parte del programa. Por ejemplo, podría incluir, en todos los progra-

mas que escriba, una primera línea que dijese como se titula dicho programa:

```
1Ø REM Título del programa
```

y el REM indicaría al Spectrum que todo lo que sigue, hasta la próxima sentencia, no debe ser tomado como una línea de BASIC y que, por tanto, no debe prestarle ninguna atención; solamente constituye una explicación para cualquier persona que pudiese leer el programa.

Podría usted pensar que es extraño que un comando tan simple haya sido dejado para el último capítulo de un libro sobre BASIC. El motivo de ello es que, aunque REM es un comando elemental, puede ser utilizado con muy buenos resultados para escribir programas que estén claros. Después que haya superado la dificultad inicial que presenta escribir programas en BASIC, debería buscar formas de escribir mejores programas. Al principio, un programa que funciona es ya una recompensa por sí mismo, pero después, su objetivo principal debería de ser conseguir programas bien escritos.

Los elementos que constituyen un programa bien escrito, los irá descubriendo por sí mismo a medida que aprenda el tipo de programación de "prueba y error" y cuando lea programas de otras personas.

La sentencia REM es parte de una programación mejor, ya que, aunque es innecesaria, ayuda verdaderamente a que sus programas sean más fáciles de comprender.

Es una buena idea incluir sentencias REM que expliquen qué es lo que sucede en cada sección de su programa, a medida que lo escribe. Las buenas explicaciones le ayudarán a entender más rápidamente sus propios programas, cuando vuelva a leerlos después de un tiempo, y servirán de auxilio a otras personas, a las que usted puede haber dado copias de dicho programa, para saber qué es lo que usted pretendía hacer en cada etapa.

Después de este libro

Como ya hemos dicho muchas veces a lo largo de este libro, el camino real para aprender a programar es escribir programas. Ciertamente, los libros le pueden servir de ayuda, pero solamente si usted está dispuesto a experimentar por sí mismo lo que aprenda. No se preocupe si sus primeros programas no hacen nada demasiado espectacular. Al principio, es mejor probar sus ideas en rutinas simples y cortas. Si intenta hacer algo demasiado complicado, hay muchas más probabilidades de que cometa errores que luego no pueda encontrar en el programa. Intente escribir programas muy reducidos que solamente hagan unas pocas cosas a la vez (si mira de nuevo el libro, verá montones de ejemplos de ese tipo).

Cuando todos sus pequeños programas funcionen, será el momento de empezar a juntarlos para construir otros más extensos.

Lo principal es seguir adelante y divertirse con su Spectrum.

Otras lecturas sobre el Spectrum y el BASIC

Una vez leído este libro, debería ser razonablemente eficaz escribiendo programas; pero, de algún modo, esto es sólo el principio.

Es de suponer que este libro le dejará con ganas de aprender más (acerca de BASIC, del Spectrum y de los computadores en general). Ahora mismo se están publicando verdaderas cantidades de libros sobre estos temas, y, por ello, todo lo que deseo hacer aquí es mencionar algunos que conozco y que, creo, podría gustarle leer.

En lo que respecta a BASIC, incluiré un título, que empieza donde termina este libro; "The Complete Programmer" (El Programador Completo), de Mike James, publicado por Granada en 1983. Después que domine los fundamentos de BASIC, puede también aprender mucho en los libros de programas de otras personas. Estos pueden usarse como un manantial de ideas y de inspiración. Son particularmente interesantes si incorporan detalles de programación y consejos útiles.

Mike James, Kay Ewbank y yo mismo, hemos colaborado para escribir esa clase de libro, precisamente: "Spectrum: Libro de juegos", que contiene algunos juegos divertidos, cuya programación resultó ser un reto. También está publicado por Díaz de Santos.

Volviendo al Spectrum, si aún no ha leído: "ZX Spectrum. Como obtener el máximo rendimiento", de Ian Sinclair (publicado por Díaz de Santos, 1983), ¡no deje de hacerlo!

OTROS TITULOS SPECTRUM

ZX SPECTRUM. COMO OBTENER EL MAXIMO RENDIMIENTO

Sinclair, I.

SPECTRUM. LIBRO DE JUEGOS

James, M.

EL PROGRAMADOR DE SPECTRUM

Gee, S. N.

SPECTRUM. INTRODUCCION AL CODIGO MAQUINA

Sinclair, I.

40 JUEGOS EDUCATIVOS PARA EL SPECTRUM

Apps, V.

OBTENGA EL MAXIMO RENDIMIENTO DE SU ZX MICRODRIVE

Sinclair, I.

GRAFICOS Y SONIDO EN EL SPECTRUM

Money, S.

DIAZ DE SANTOS, S.A.

MADRID-BARCELONA